ProviewR

OPEN SOURCE PROCESS CONTROL

Handbok för konstruktörer

2024-02-25

Version 6.1.5

2 Inledning

ProviewR är ett modernt, kraftfullt och generellt process styr system. Det innehåller alla funktioner som normalt krävs för effektiv sekvensstyrning, reglering, datainsamling, kommunikation, övervakning mm.

Konfigureringen av ett ProviewR system görs grafisk, vilket gör att anpassningen till en tillämning blir enkel, pålitlig och flexibel. ProviewR är ett distribuerat system, vilket innebär att systemet består av flera datorer sammankopplade i ett nätverk. Via nätverket utbyter datorerna ata med varandra. På det här sättet blir, t ex uppmätta signalvärden, åtkomliga på alla process- och operatörsstationer i ett ProviewR system.

Ett ProviewR system är definierat med objekt. Varje objekt representerar en fysisk eller abstrakt enhet i systemet. Objekten ordnas i en hierarktisk trädstruktur, vilket gör det möjligt att ge en strukturerar beskrivning av systemet. Trädstrukturen influerar också en hierarktisk namngivning av alla objekt. Genom att noggrant välja namn vid konfigureringen av ett system, kommer det fullständiga objektsnamnet att identifiera dess funtion, eller roll i systemet.

Hierarkierna är uppdelade i två grupper, en kallas anläggningshierarkin, och beskriver systemet ur ett logiskt perpektiv. Den andra kallas nodhierarkin, och beskriver systemet ur ett fysiskt perspektiv. Konfigureringen av dessa olika delar kan göras oberoende av varandra. Slutligen kopplas de två delaran ihop.

För att konfigurera ett system använder man ProviewR's arbetsbänk. Arbetsbänken omfattar en permanent databas och ett antal verktyg för att konfigurera de objekt som behövs för att konfigurera ett system. Utifrån arbetsbänken skapar man ett körbart system, liksom dokumentation om systemet.

Avsikten med ProviewR är att hjälpa dig att skapa automatierarde system. Antag att du har en process som du vill styra, ProviewR hjälper dig att skapa styrsystemet för processen, och när systemet är skapat, finner du att du även har gjort dokumentationen för systemet.

3 Översikt

Eftersom det här är en handledning för konstruktörer kan det vara lämpligt att inleda med en beskrivning av vad designen av ett styrsystem innebär. Beskrivningen kan även fungera som en introduktion till de olika begrepp som kommer att användas i den här handledningen.

En konstruktör utgår naturligtvis från den process som styrsystemet ska styra, och den första uppgiften är att lära sig processen och fundera över bästa sättet att styra den: vilka reglerloopar som behövs, vilka förreglingar som ska finnas, hur anläggningen startas upp och stoppas, hur operatörer och underhållare ska arbeta mot systemet. Detta sammanfattar man i en Funktions specification.

Samtidigt måste man fundera på vilken information om processen som styrsystemet behöver för att kunna utföra sin uppgift, dvs vilka givare som ska placeras i anläggningen. Styrsystemet måste även kunna påverka processen genom på olika sätt, t ex mha ventiler och motorer. Detta resulterar i en Signal lista, som är lista på alla in och utgående signaler till systemet.

I det här läget dyker även frågan upp om vilket styrsystem man ska använda sig av, och ett alternativ är naturligtvis ProviewR. Man måste även bestämma sig för vilket I/O-system man ska använda, och hur man ska dela upp funktionen i olika processstationer.

I/O-system

I/O-systemet uppgift är att ta in signaler från processen till styrsystemet, och att ställa ut signaler för att påverka processen. Signalerna är vanligtvis digitala eller analoga, men det finns även andra typer som heltalsvärden och pulsgivare. Man kan välja mellan rack och kort system i anslutning till datorn, eller distributerat I/O som t ex profibus.

Konfigurering

När det är dags att börja konfigurera system, skapar man först ett nytt Projekt i Administratören. Administratören ett verktyg för skapa ordning och reda bland alla projekt, eftersom dessa kan bli ganska många med tiden.

Konfigureringen av ett system sker till stor del genom att skapa objekt i en databas, arbetsbänken. Det finns en stor mängd olika objekt för att konfigurera allt från IO-kanaler till processbilder. ProviewR's objektshandbok innehåller över 800 typer av objekt. Objekten läggs upp i en trädstruktur och man använder ett verktyg som kallas för Konfiguratören för att skapa objekt och för att navigera i objektsträdet.

Objektsträdet delas in i två delar, anläggningshierakin och nodhierakin. Anläggningshierakin speglar olika funktioner i anläggningen och processen, medan nodhierakin speglar styrsystemets uppbyggnad ur hårdvaru synpunkt med datorer, I/O-rack och I/O-kort.

När man senare startar upp styrsystemet i runtime, skapas en kopia av objektträdet som läggs in i en realtidsdatabas, rtdb. Överföringen från arbetsbänken till rtdb sker med s k laddatafiler, filer som genereras från arbetsbänken och som innehåller alla objekt som finns i denna.

Styrprogram

ProviewR innehåller ett grafisk programmeringsspråk med vilket man programmerar logik, grafcet-sekvenser och reglerkretsar. Det går under benämningen PLC program. Även PLC programmet ingår som en del av objektträdet. Det konfigureras genom att man placerar ut speciella program objekt, PlcPgm, i anläggninghierarkin. När man öppnar ett PlcPgm kommer man in PlcEditorn, i vilken man gör den grafiska programmeringen. Här skapas funktionsobjekt som binds samman i ett signalflöde av digitala och analoga signaler, där ingångssignaler hämtas upp på vänstersidan, tranformeras i olika funktionsblock för att slutligen ställas ut till utsignaler på högersidan.

Ett komplement till PLC programmet är applikations-program, som skrivs i c, c++ eller java. Applikationer skrivs och startas som fristående program och knyter upp sig mot realtidsdatabasen mha ett API.

Simulering

Realtidsdatabasen, PLC-programmet och eventuella applikationer kan enkelt startas upp på utvecklingsstationen. Det här gör att man kan testa sina progam i direkt anslutning till programmerandet. Man kan även skriva speciella simulerings-program som läser utgångssignaler, simulerar den påverkan utgångarna har på processen, beräknar värden på olika givare och sätter dessa värden i ingångssignaler.

Konfigureringen och programmeringen av systemet blir då en process, där man växelvis konfigurerar/programmerar och testar. Resultatet blir väl avlusade program och en snabb och effektiv igångkörning av anläggningen. Det leder även till bättre program och mer genomarbetade funktioner, eftersom återkopplingen blir större i den kreativa process som konstruktionen av ett styrsystem innebär.

Vid simuleringen och igångkörning är det av yttersta vikt att ha tillgång till verktyg gör att man kan övervaka och undersöka systemet, och snabbt lokalisera eventuella fel. I ProviewR kallas det här verktyget Xtt. Xtt innehåller en mängd funktioner för undersöka innehållet i realtidsdatabasen, för att följa signalflöden, logga snabba eller långsamma förlopp, etc.

Operatörs gränssnitt

Det finns en rad olika yrkesgrupper som ska kunna komma åt systemet, operatörer som sköter den dagliga driften, underhållare som dyker upp när något har gått fel, process ingenjörer som vill ha ut process data av olika slag. Alla har olika krav på gränssnitt mot systemet. Dessutom kan gränserna mellan olika yrkesgrupper vara flytande, operatörer som är både operatörer och underhållare, och kanske även processtekniker. Detta ställer stora krav på funktionalitet och flexibilitet i operatörsgränssnitten.

ProviewR innehåller en operatörsmiljö där processbilder, felsökningsverktyg, presentation av trendkurvor, datablad, hjälptexter, larmlistor mm är väl integrerade och länkade så att den rätt konfigurerad, blir ett oerhört effektivt hjälpmedel för alla användare. Man kan snabbt och enkelt genom s k metoder, som aktiveras från popupmenyer, hämta upp all information som finns om olika objekt, i realtidsdatabasen eller på olika serversystem, i form av PLC kod, trendkurvor, datablad mm.

Processbilder

Processbilder byggs i en grafisk editor (Ge). Grafiken är vektorbaserad, vilket gör att alla bilder och komponenter kan skalas obegränsat. Komponenter har en förprogrammerad dynamik för att ändra färg och form beroende på signaler i realtidsdatasen, eller reagera på musklick och sätta värden i databasen. På varje komponent som är känslig för musklick eller inmatning kan man ange behörighet, och selektivt tillåta, eller hindra, användare att påverka systemet.

Övervakning

Om något fel uppstår i processen, måste operatören uppmärksammas på detta. Det sker med speciella övervaknings-objekt, som konfigureras i anläggningshierakin eller i PLC-programmet, och som ger upphov till larm eller meddelanden. Larmen har fyra prioritets nivåer: A, B, C eller D, och presenteras för operatören i larmlistan, händelselistan och den historisk händelselistan.

Larmlistan inhåller okvitterade och rådande larm. Ett larm måste normalt kvitteras av operatören innan de försvinner från listan. Om larmtillståndet fortfarande är rådande, ligger larmet kvar i listan så länge det är rådande.

Larm registreras även i händelselistan, som presenterar händelser som inträffat i kronologisk ordning.

Historiska händelselistan är en databas som även den registrerar händelser. Här kan man söka efter larm med olika kriterier som prioritet och anläggningsdel.

Om en anläggningsdel ställs av kan man blockera larmen från den, så att operatören inte distraheras av larm utan betydelse. Blockerade anläggningsdelar visas in en Blockerings lista.

Datalagring

Ofta vill man kunna se hur en signal förändras över tiden, i form av en kurva. I ProviewR finns tre olika funktioner för detta, DsTrend, DsFast och SevHist.

DsTrend är en trendkurva som lagras i realtidsdatabasen. Mätvärdet för en signal lagras kontinuerligt med ett intervall på 1 sekund och uppåt. För varje kurva finns det plats för ca 500 mätvärden, så väljer man att lagra ett nytt värde varje sekund får man en kurva på hur signalen har förändrats under ca 8 min.

SevHist lagrar signaler på liknande sätt i en databas på disk, vilket gör att man kan lagra värden under längre perioder än DsTrend.

DsFast lagrar ofta snabbare förlopp, där lagringen startas på ett triggvillkor, och forsätter under en specificerad tid, för att sedan presenteras i kurvform.

4 Databas struktur

Som vi har sett tidigare så sker största delen av konfigureringen av ett ProviewR system i en databas, Arbetsbänken. I Arbetsbänkan skapar man objekt i en träd-struktur, och varje objekt ger upphov till en viss funktion i styrsystemet. ProviewR är vad man brukar kalla objekts orienterat, och låt oss titta lite närmare på vad ett ProviewR-objekt egentligen är.

4.1 Objekt

Ett objekt består av en datamängd, som på något sätt definierar objektets tillstånd eller egenskaper. Datamängden kan vara mycket enkel, som t ex hos en Och-grind, där den består ett boolskt värde som kan vara sant eller falskt. En PID regulator däremot, har en mera komplex datamängd. I denna finns förstärkning, integrationstid, utsignal, tvångsstyrning mm. Den består av en blandning av digitala, analoga och heltalsvärden. Vissa värden konfigureras i utvecklingsmiljön, medan andra beräknas i runtime.

Datamängden kallas objektets kropp. Kroppen är indelad i attribut, och varje attribut har ett namn och en typ. Kroppen för en Och-grind består av attributet Status som är av typen Boolean, medan kroppen för en PID regulator består av 47 st attribut: ProcVal, SetVal, Bias, ForceVal etc.

Alla PID objekt har sin datamänd strukturerad på samma sätt, och man säger att de tillhör samma klass. PID objekten tillhör klassen PID och Och-grindarna tillhör klassen And. En klass är en slags mall för hur objekt som tillhör klassen ska se ut, t ex vilka attribut som ingår, och attributens namn och typ.

Förutom en kropp, har ett objekt även ett huvud. I huvudet ligger objektets klass, identitet, namn och relation till andra objekt. Objekten är inordnade i en trädstruktur, och i huvudet finns länkar till objektets förälder och närmaste syskon.

4.2 Volymer

När man konfigurerar ett system och skapar objekt, vet man vanligtvis vilken nod objektet ska tillhöra i runtime. Man skulle kunna dela in objekt efter vilka noder de kommer att tillhöra, men man har valt en lite mer flexibel indelning, så istället delar man in objekten i volymer. En volym är en slags behållare för objekt. Volymen har ett namn och en identitet, och den innehåller ett antal objekt ordnade i en trädstruktur.

Det finns ett antal olika typer av volymer, och den första man kommer i kontakt med är en rotvolym. När man konfigurerar en nod, jobbar man vanligtvis i en rotvolym. Varje nod är kopplad till en rotvolym, vilket innebär att när noden startas i runtime, kommer rotvolymen, och de objekt som finns in den, att laddas in i noden. Nedan följer en beskrivning på de olika typer av volymer som finns.

RootVolume

En rotvolym innehåller roten till objektsträdet på en nod. När en nod startar, laddar den in rotvolymen.

En nod är kopplad till en och endast en rotvolym. Däremot kan en rotvolym laddas in i flera olika noder. Samtidigt som man kör processtationen i produktion, kan man ladda in samma rotvolym i sin utvecklingsstation för simulering, och i en tredje nod för utbildning.
Man måste dock se till att de olika noderna går på olika kommunikations bussar.

SubVolume

En del av objekten på en nod kan man lägga i en subvolym. Anledningen till att dela upp objekten i en nod i en rotvolym och en eller flera subvolymer, kan vara att flera personer måste konfigurera noden samtidigt, eller att man planerar att flytta styrningen av vissa anläggningsdelar till en annan nod så småningom.

ClassVolume

Definitionen av olika klasser ligger speciella volymer som kallas klassvolymer. Här byggs beskrivningen av en klass upp med objekt som definierar klassens namn och vilka attribut som ingår i klassen.

Det finns två stycken klassvolymer som man alltid måste ha med i ett ProviewR system, pwrs och pwrb. pwrs innehåller systemklasser, framför allt klasser som används i klassdefinitionerna. pwrb innehåller basklasser, dvs standard klasser som behövs för att bygga en process- eller operatörsstation.

DynamicVolume

En dynamisk volym innehåller dynamiska objekt, dvs objekt som skapas temporärt i runtime. Om man har en materialföljnings modul i systemet, kan man skapa ett objekt för varje material som behandlas i anläggningen, och ta bort det när materialet är färdigbehandlat.

SystemVolume

Systemvolymen är en dynamisk volym som finns i varje nod, och som innehåller diverse systemobjekt.

DirectoryVolume

Directory volymen är en volym som enbart finns i utvecklingsmiljön. I denna konfigurerar man vilka volymer och noder som finns i systemet.

Volymsidentitet

Varje volym har en unik identitet, som skrivs med fyra tal, separerade med punkter, t ex "_V0.3.4.23". Prefixet _V markerar att det är frågan om en volymsidentitet. För att verifiera att volymsidentiteter är unika finns det en global volymslista som innehåller alla volymer. Innan man skapar ett projekt, ska volymerna i projektet registreras i volymslistan.

4.3 Attribut

Datamängden i ett objekt är indelat i attribut. Varje attribut har ett namn och en typ.
Här följer en beskrivning på de vanligaste attribut typerna.

Boolean

Digitala attribut är av typen boolean, som kan ha värdet sant (1) eller falskt (0).

Float32

Analoga attribut är av typen Float32, dvs ett 32-bitars flyttal.

Int32

Heltals attribut är vanligen av typen Int32, dvs ett 32-bitars heltal. Men det finns även ett antal andra typer av heltal: Int8, Int16, Int64, UInt8, UInt16, UInt32 och UInt64.

String

I ett string-attribut kan man lagra en sträng av ascii-tecken. Det finns olika sträng typer för olika längd på strängen, t ex String8, String16, String40, String80 och String256.

Time

Time innehåller en absoluttid, t ex 1-MAR-2005 12:35:00.00.

DeltaTime

DeltaTime innehåller en tidsskillnad, t ex 1:12:22.13 (1 timme, 12 minuter, 22.13 sekunder).

Enum

Enum är en uppräkningstyp, som används när man ska välja ett av ett antal alternativ. Den kan anta ett antal heltalsvärden, där varje värde är kopplat till ett namn. Det finns t ex en ParityEnum som kan ha värdet 0 (None), 1 (Odd) eller 2 (Even).
Enum är här en bastyp och ParityEnum en härledd typ.

Mask

Mask används när ska välja ett, eller flera av ett antal alternativ. Alternativen representeras av bitarna i ett 32-bitars heltal.

Ett attribut kan även bestå av en mer komplex datastruktur. Det kan vara en vektor med ett visst antal element, och det kan faktisk också vara ett annat objekt, ett s k attributobjekt.

4.4 Klass

En klass är en beskrivning på hur ett objekt som tillhör klassen ska se ut. Ett objekt som tillhör klassen kallas för en instans. I klassen definieras hur instansernas datamängd är strukturerad i attribut av olika typ, eller hur objektet grafiskt ska representeras i PLCeditorn eller i operatörsmiljön.

Varje klass har ett template objekt, en instans av klassen som innehåller defaultvärden för olika attribut i klassen.

ProviewR's bassystem innehåller ca 1000 klasser. Se Objektshandboken för närmare beskrivning. Konstruktören kan även skapa egna klasser inom ett projekt.

4.5 Objektsträd

Objekten i en volym är ordnade en trädstruktur. I volymen finns ett eller flera topp objekt, varje topp objekt kan ett eller flera barn, vilka i sin tur kan ha barn, osv. Man brukar tala om relationerna mellan objekt i trädet i termer av förälder, syskon, barn, förfader och ättlingar.

4.6 Objektsnamn

Varje objekt har ett namn som är unikt inom sin syskonskara. Objektet har även ett fullständigt namn som är unikt i världen. I det fullständiga namnet ingår, förutom objektsnamnet, även namnet på volymen och på samtliga förfäder, t ex

```
VolTrafficCross1:TrafficCross1-ControlSignals-Reset
```

Vill man vara ännu mer specifik, och peka ut ett attribut i ett objekt, hakar man på attributnamnet på objektsnamnet med en punkt emellan, t ex

```
VolTrafficCross1:TrafficCross1-ControlSignals-Reset.ActualValue
```

Även ett attribut kan ha flera namnled, eftersom ett attribut kan bestå av ett objekt. Attribut namnleden separeras då av punkter, t ex

```
VolTrafficCross1:Hydr-Valve.OpenSw.ActualValue
```

4.7 Montering

En operatörstation ska kunna presentera värden på signaler och attribut som ligger i processtationernas volymer. Detta åstadkommer man genom att operatörsstationen monterar processnodernas volymer i det egna objektsträdet. En montering innebär att man hänger in ett objektträd från en annan volym i den egna rotvolymen. Var i trädet volymen ska hängas in konfigureras med ett MountObject objekt. I MountObject objektet anges vilket objekt i den andra volymen ska monteras. Resultatet blir att MountObject objektet presenteras som det monterade objektet, inklusive det objektträd som ligger under. Skenbart ser det ut som om objektet tillhör den egna rotvolymen, medan de i själva verket ligger på en annan nod.

Om man använder sig av subvolymer måste även dessa monteras i en rotvolym för att objekten ska bli tillgängliga.

När man väljer monteringspunkter och namn på monteringsobjekt, är det lämpligt att göra detta så att objektet får samma hierakinamn i båda volymerna.

4.8 Objektsidentitet

Ett objekt har en identitet som är unik. Den består av volymsidentiteten och ett objektindex som är unikt inom volymen. En objektsidentitet skrivs t ex "_O0.3.4.23:34" där 0.3.4.23 är volymsidentiteten och 34 objektindex. Prefixet _O markerar att det är en objektsidentitet.

5 En fallstudie

I det här kapitlet ska vi ge en idé om hur ett ProviewR system skapas. Processen som ska styras i det här fallet är mycket enkel, en korsning mellan fyra trafikljus, men kommer att ge ett begrepp om de olika steg man ska genomgå för att skapa ett ProviewR system.

Trafikljuset ska kunna arbeta i två olika moder:

- Normal: trafikljusen går i den normala cykeln röd, gul och grön.
- Blinkande: trafikljusen blinkar gult.

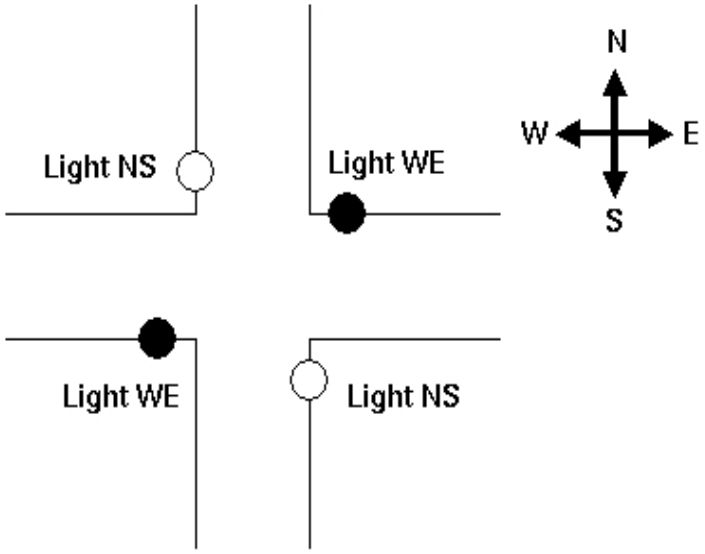


Fig Trafikljusen i en korsning

Arbetsmoden för trafikljuset bestäms av en operatör via en operatörsstation, eller av en underhållstekniker som påverkar ett vred. Underhållsteknikern kan ändra mod endast om operatören har ställt trafikljusen i service mod.

Figuren 'Trafikljus, kontrollpanel' visar de olika vred och indikatorer som krävs av operatören resp underhållsteknikern, för att kunna övervaka och styra systemet. Dessa kan implementeras som processbilder på operatörsstationen eller med hårdvara.

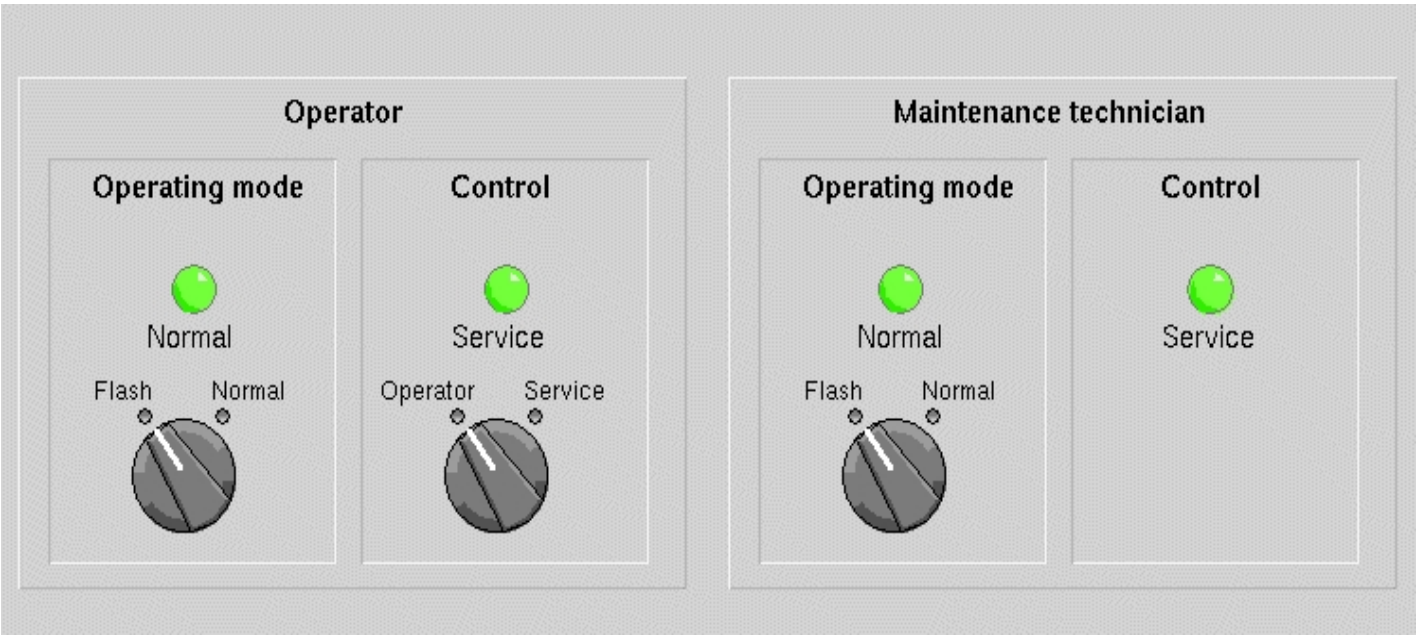


Fig Trafikljus, kontrollpanel

5.1 Specifikation av I/O

Vi börjar med att analysera processen för att bestämma vilken hårdvara vi ska använda.

Digitala utgångar

Vi har fyra trafikljus, men trafikljusen på samma gata kan parallellkopplas, vilket betyder att vi kan behandla dem som två ljus.

Tre utgångar per ljus: 2 * 3 = 6
Indikator: Operating mode 1
Indikator: Control 1

Total antal digitala utgångar: 8

Digitala ingångar

Den enda digitala ingång som behövs är för underhållsteknikerns vred. Operatören styr processen från sin operatörsstation och kräver inga fysiska signaler.

Omkopplare: Operating mode 1

Totalt antal digitala ingångar: 1

Analogt I/O

Analoga in- eller utgångar behövs inte i det här fallet.

Specifikation av processstationen

När vi har bestämt det I/O som krävs, kan vi välja hårdvara för processstationen. Vi väljer:

1 Linux PC med rack.

1 kort med 16 digitala ingångar.
1 kort med 16 digitala utgångar.

Specifikation av operatörsstationen

1 Linux PC.

Specifikation av process grafik.

Vi behöver en skärm från vilken operatören kan styra och övervaka trafikljusen.

5.2 Administration

Först måste vi registrera en ny volym, skapa ett projekt, och eventuellt skapa nya användare. För detta krävs:

- Ett namn för projektet. Vi kallar det trafficcross1.
- Två volymer, en för processtationen och en för operatörsstationen.
- Vi behöver tre användare: en utvecklare, en operatör och en underhållstekniker.

Volymer, projekt och användare skapas och registreras med 'administratören'.

Registrera volymer

För det här projektet behövs två volymer, en för processtationen, och en för operatörsstationen. De är rotvolymer så vi kan välja lediga volymsidentiteter i intervallet 0.1-254.1-254.1-254. Vi väljer 0.1.1.1 till processtationen och 0.1.1.2 till operatörsstationen, och går in i volymsmod i administratören för att registrera volymerna.

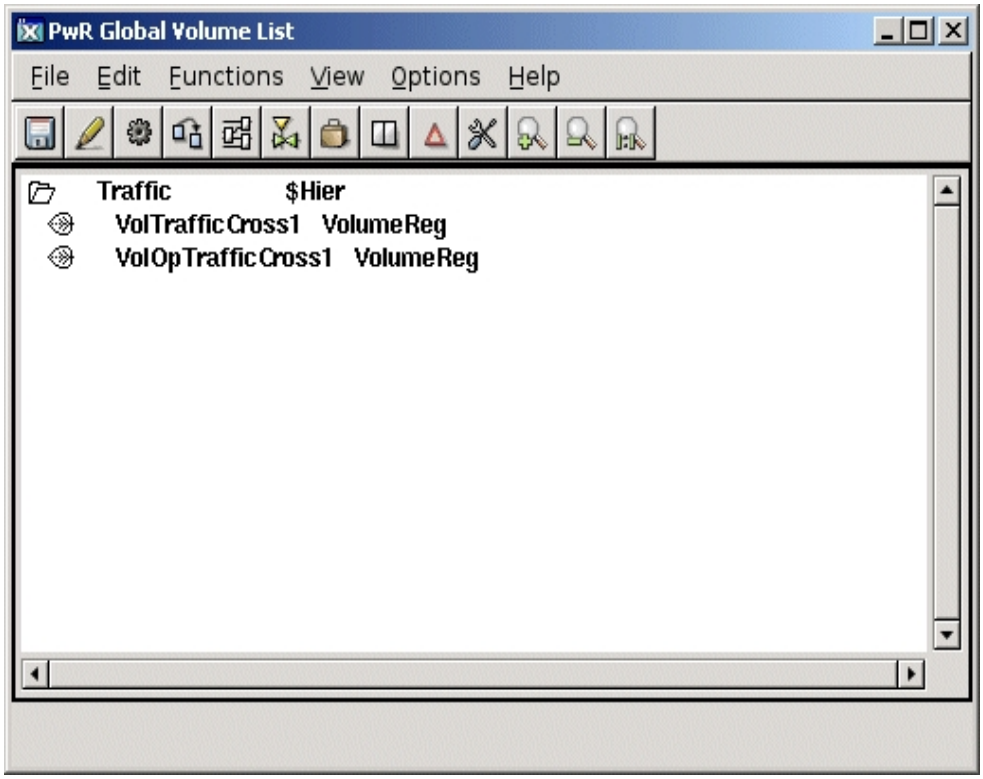


Fig Registrering av volymer

Användare

Eric är utvecklare vid trafikavdelningen, Carl är operatör och Lisa är underhållstekniker. De är alla inblandade i samtliga projekt på trafikavdelningen, så vi skapar en gemensam systemgrupp för all projekt, och låter dem dela användare. Vi tilldelar Eric utvecklings-privilegier, Carl operatörspriviligier och Lisa underhållspriviligier.

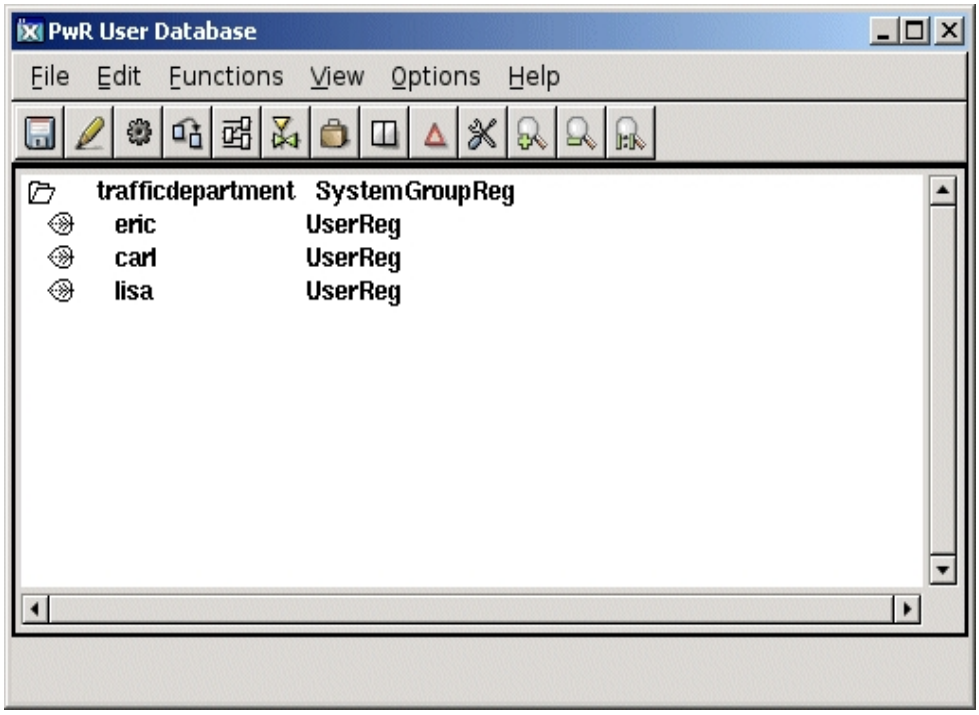


Fig Skapade användare

Skapa projekt

Vi skapar projektet med hierarkinamnet traffic-trafficcross1

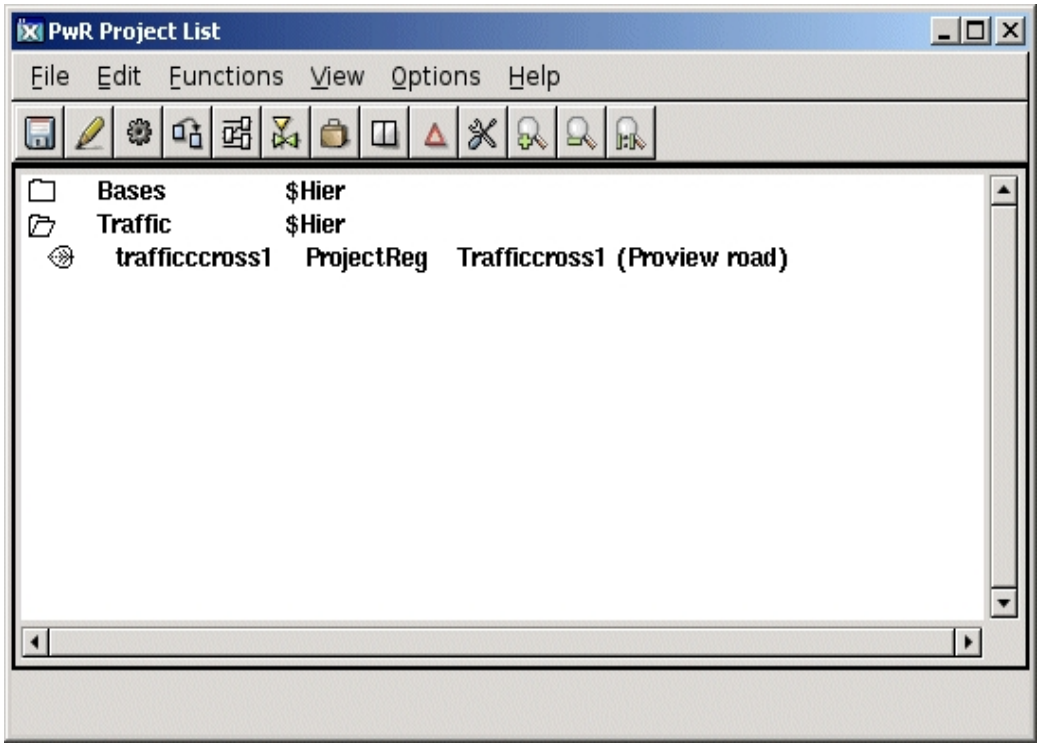


Fig Skapat projekt

Konfigurera projektet

Projektet har en direktoryvolym där noder och volymer i projektet ska konfigureras. I det övre fönstret konfigureras volymerna med RootVolumeConfig objekt. I det under fönstret konfigureras process- och operatörsstationerna med NodeConfig objekt. NodeConfig objektet läggs under ett BusConfig objekt som anger vilken QCOM bus noderna ska kommunicera på.

NodeConfig objekten innehåller

- nodenamn
- ip adress för noden

Under varje NodeConfig objekt finns ett RootVolumeLoad objekt som anger vilken volym som ska laddas in när runtimemiljön startas.

Notera också systemobjektet med attributet SystemGroup som tilldelas värdet "trafficdepartment". Detta ger användarna eric, carl och lisa tillgång till projektet.

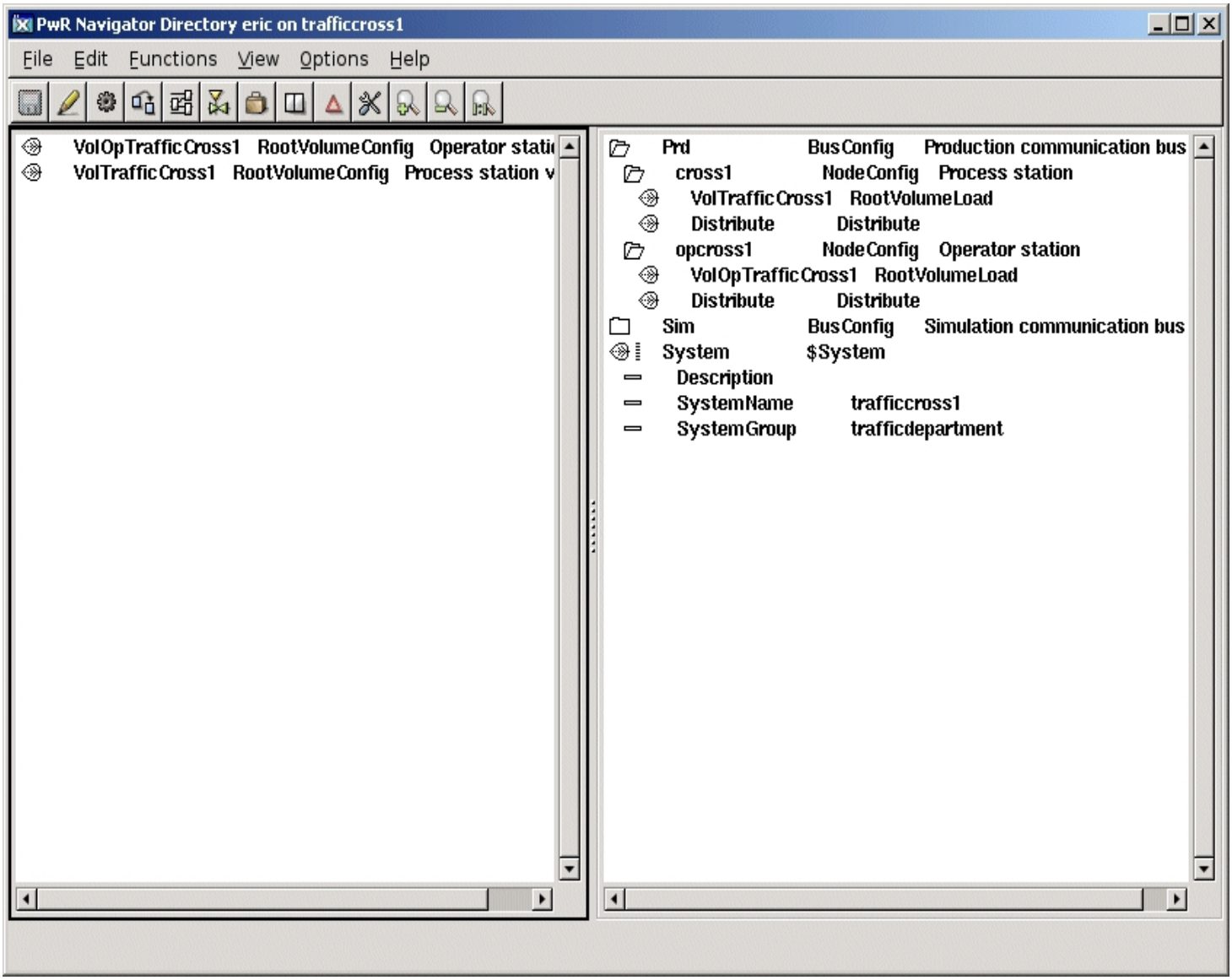


Fig Directory volymen

5.3 Konfigurering av anläggninghierarkin

När konfigureringen av projektet är klar, kan vi börja ägna oss åt själva anläggningen. Konfigureringen av anläggningen görs i Konfiguratören. Anläggningshierarkin är en logisk beskrivning av verkligheten som ska styras och övervakas.

Processstationen

Den största delen av konfigureringen görs i processstationens volym, VolTrafficCross1. Det är ju här som all kort, kanaler, signaler och PLC program ska konfigureras.

Anläggningshierarkin har en trädstruktur. Exempel på nivåer i hierarkin kan vara anlägging, process, processdel, komponent och signal. Signaler är de logiska signaler som representeras av signalobjekt, vilka kopplas till fysiska kanaler.

Ibland kan det vara svårt att konfigurera varje signal i ett inledande skede, men men måste ändå bestämma hur tänkbara signaler ska grupperas.

Figuren nedan visar hur en anläggninghierarki har konfigurerats. Vi ser hur signaler ligger på olika nivåer, och också hur Plc program är konfigurerade i anläggningshierarkin.

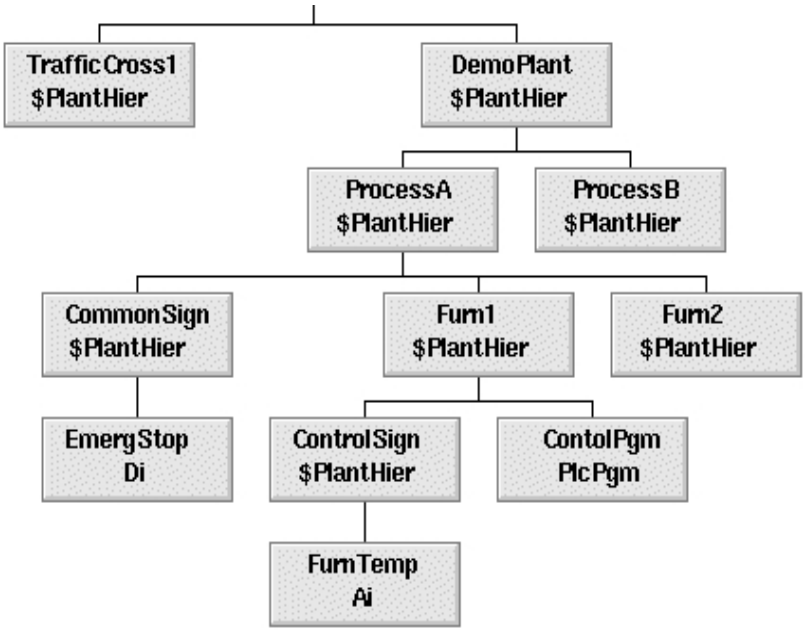


Fig Exempel på anläggningshierarki

Vi väljer att kalla vår anläggning TrafficCross1 och bestämmer följande struktur:

- Två trafikljus, var och en bestående av en grön, en gul och en röd lampa. Eftersom gatorna ligger i Nord-Sydlig resp Väst-Östlig riktning, kallar vi dem TrafficLightNS and TrafficLightWE (West-East). Varje lampa behöver en signal, av typen digital utgång som vi kallar RedNSm, RedWE etc.

- Ett PLC program som styr trafikljuset.

- Ett antal styrsignaler för att välja arbetsmod och funktion. Vi väljer att lägga dem under en map, ControlSignals. Tabellen nedan visar de signaler som krävs.

Signalnamn	Signaltyp	Funktion
ServiceSwitch	Di	En omkopplare som underhållsteknikern kan påverka för att ändra arbetsmod.
OperatorSwitch	Dv	Ett värde som operatören kak påverka för att ändra arbetsmod.
ServiceMode	Dv	Ett värde som underhållsteknikern kan påverka för att servicemod.
ServiceModeInd	Di	En signal som visar underhållsteknikern att att programmet är i servicemod.
Mode	Dv	Indikerar om programmet är i normal eller blinkande mod.
ModeInd	Do	Indikerar om programmet är i normal eller blinkande mod.
Reset	Dv	Ett värde för att återställa programmet i utgångsläge.

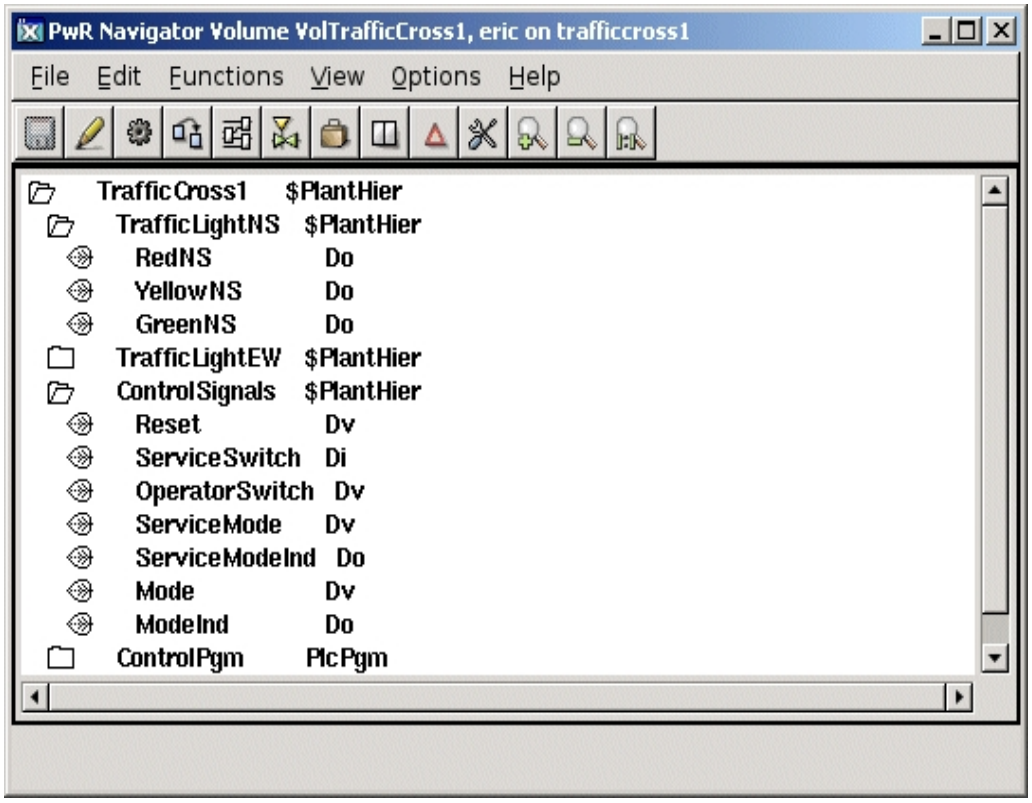


Fig Anläggningshierarkin för korsningen

På översta nivån finns ett anläggningsobjekt, TrafficCross1 av klassen $PlantHier. Vi använder andra objekt av klassen $PlantHier för att gruppera objekten. Vi skapar också ett objekt som definierar ett PLC program, ControlPgm objektet av klass PlcPgm.

Operatörsstationen

Operatörstationens konfigurering görs i volymen VolOpTrafficCross1. På anläggningssidan finns här endast ett monteringsobjekt, som gör att anläggningshierarkin i processnoden blir tillgänglig på operatörsstationen. Vi har monterat det översta $PlantHier objektet, 'TrafficCross1' med ett MountObject med samma namn.

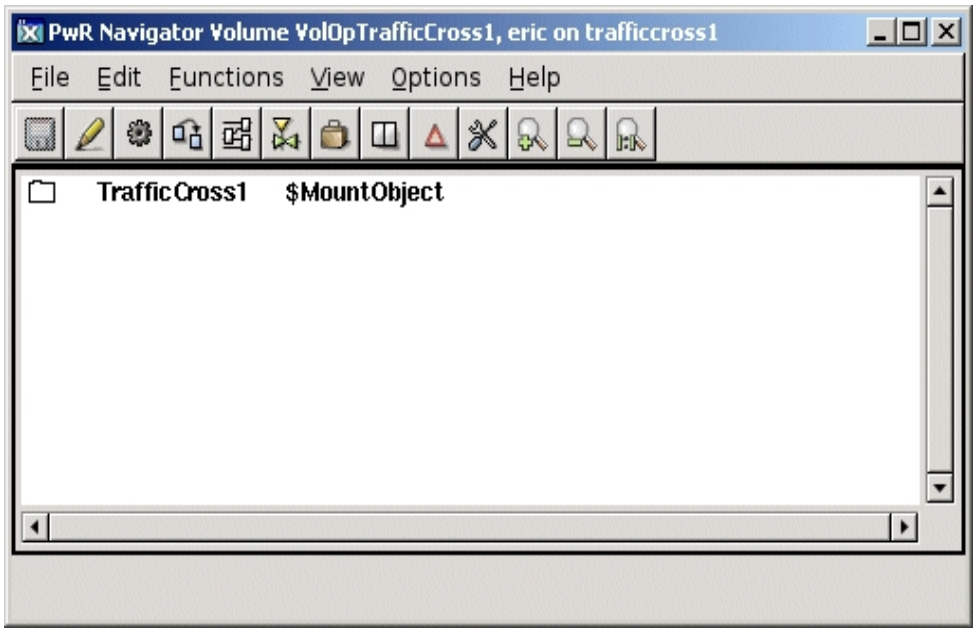


Fig Anläggningshierarkin i operatörsvolymen

5.4 Nod konfiguration

Efter anläggninghierarkin, ska nodhierakin konfigureras.

Processtation

I det här exemplet väljer vi att starta konfigureringen med processtationen. Vi benämner den "cross1". Vi rekommenderar att men ger noderna beskrivande namn. I nodehierakin skapar vi ett $NodeHier objekt 'Nodes' och under det ett $Node objekt 'cross1' som konfigurerar noden.

I anlysfasen beslutade vi att processtationen skulle bestå av följande hårdvara:

- 1 Linux PC
- 1 rack med 16 kortplatser
- 1 kort med 16 digitala ingångar (Di kanaler).
- 1 kort med 16 digitala utgångar (Do kanaler).

Rack och kort konfigureras på ett sätt som speglar den fysiska konfigureringen. Du placerar ett rack i nod objektet, korten under rack objektet, och kanalerna under respektive kort objekt.

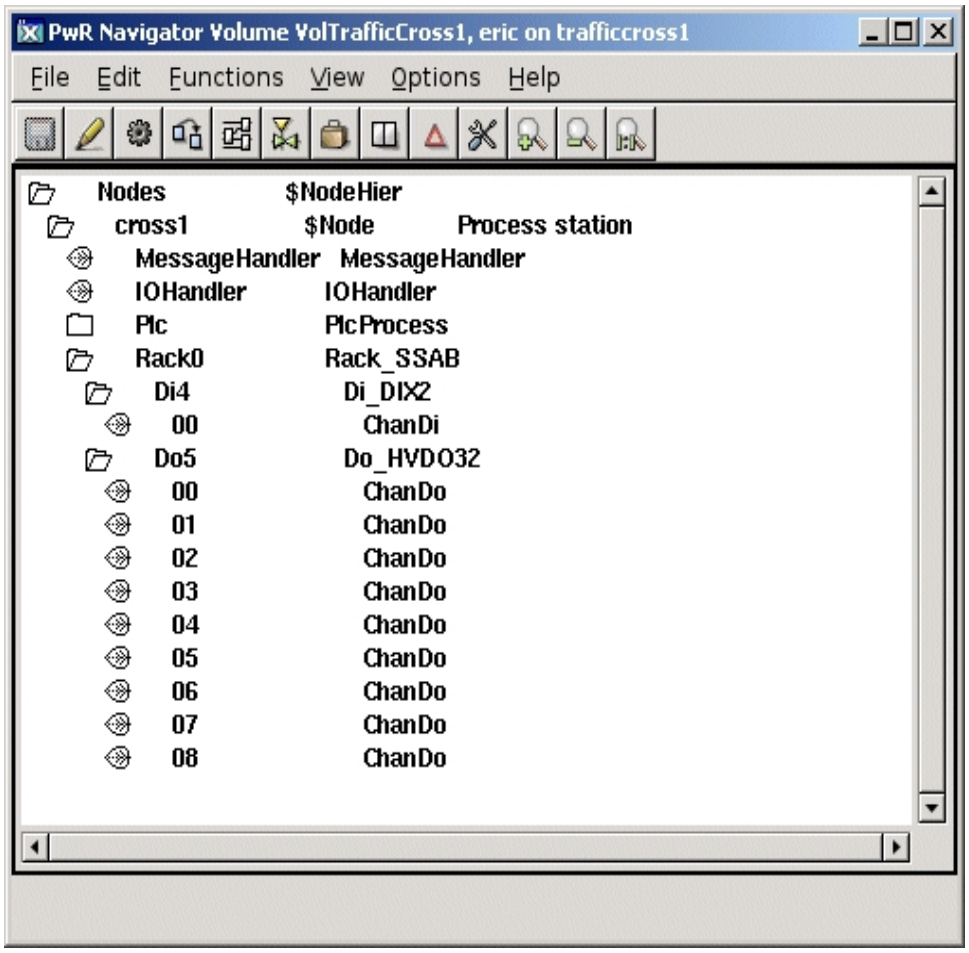


Fig Nodehierarkin i processstationen

Vi konfigurerar även PLC programmet med ett PlcProcess objekt, och under detta ett PlcThread objekt för varje tidbas. Vi nöjer oss med en tidbas på 100 ms.

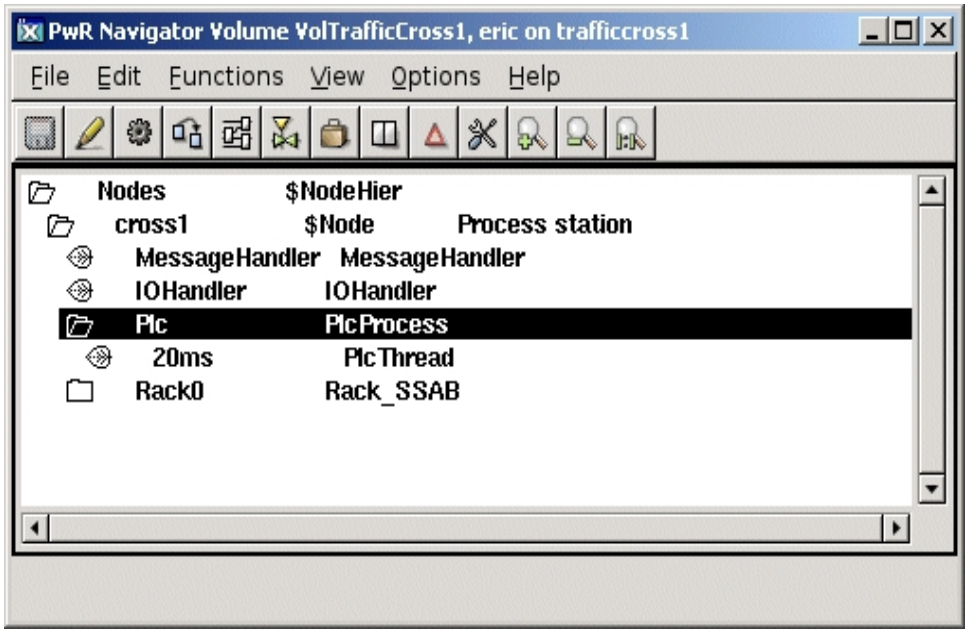


Fig Konfigurering av tidbaser i PLC programmet

Varje objekt har att antal attribut som måste datasätta. För att ge en inblick i hur detta

går till visar vi några attribut i PlcProcess objektet.

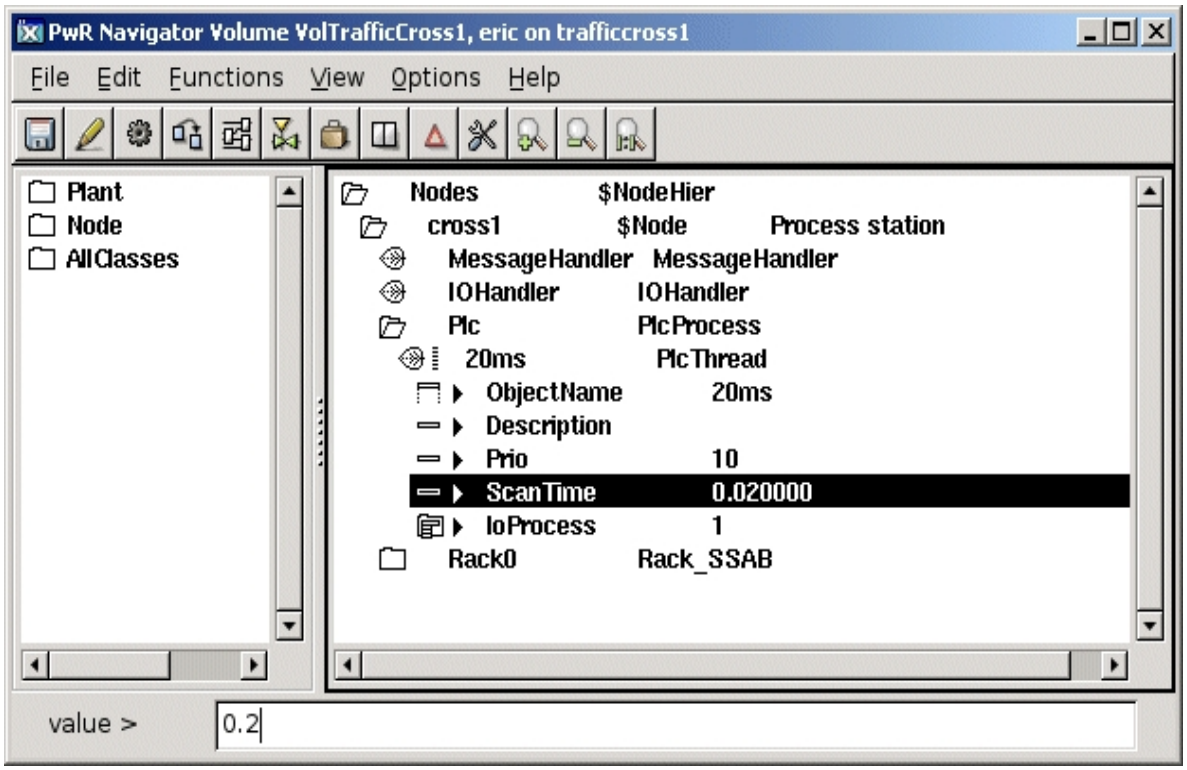


Fig Ändring av värden på attribut

Operatörsstation

Operatörsstationens nodhierarki konfigureras i volymen för denna VolOpTrafficCross1. Under nod objektet finner vi ett OpPlace objekt som definierar operatörsplatsen, och under detta ett User objekt som definierar en användare. Under OpPlace objektet finns även ett XttGraph objekt för operatörsplatsens processbild.

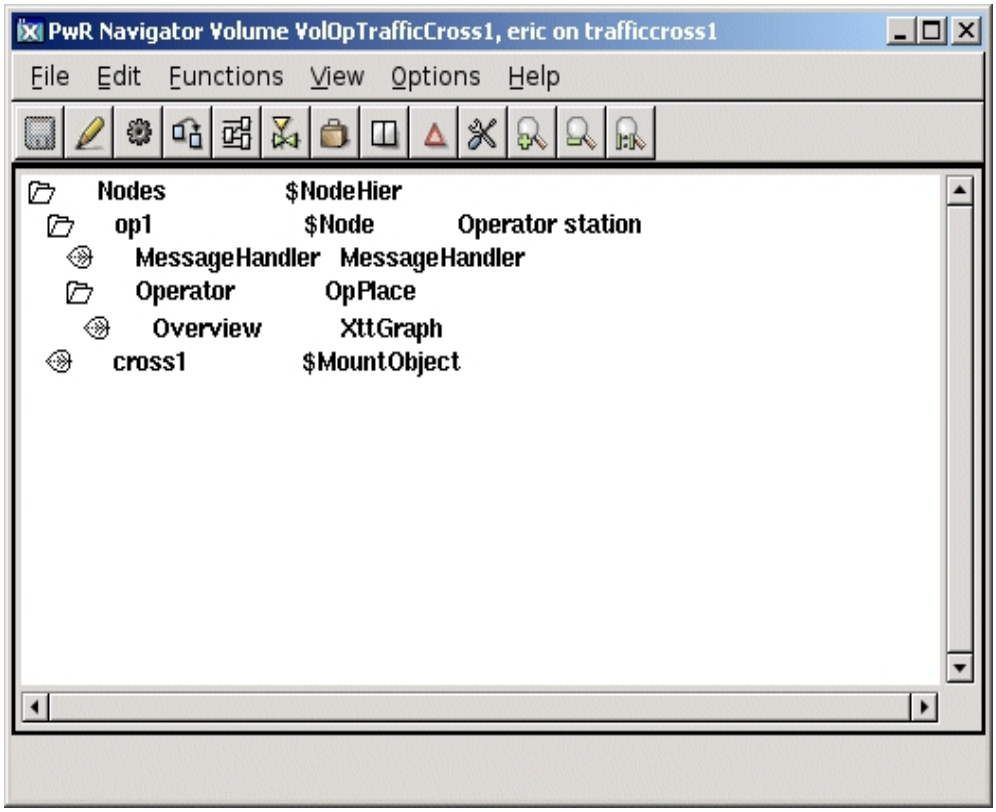


Fig Nodhierarkin i operatörsvolymen

Koppla kanaler till signaler

När både anläggnings- och nodehierarkin är konfigurerad, är det dags att koppla ihop logiska signaler med fysiska kanaler. Varje logisk signal ska kopplas till en fysisk kanal, en Di till en ChanDi, en Do till en ChanDo, en Ai till en ChanAi, en Ao till en ChanAo, etc.

Man kan se kopplingen som en representation av kopparkabeln mellan komponenten ute i anläggningen och kanalen i I/O racken. I figuren nedan finns ju en kabel mellan omkopplaren och kanal 0 på Di-kortet. Eftersom Di-signalen ServiceSwitch representerar omkopplaren, och Di-kanalen Di4-00 representerar kanalen, måste vi göra en koppling även mellan dessa objekt.

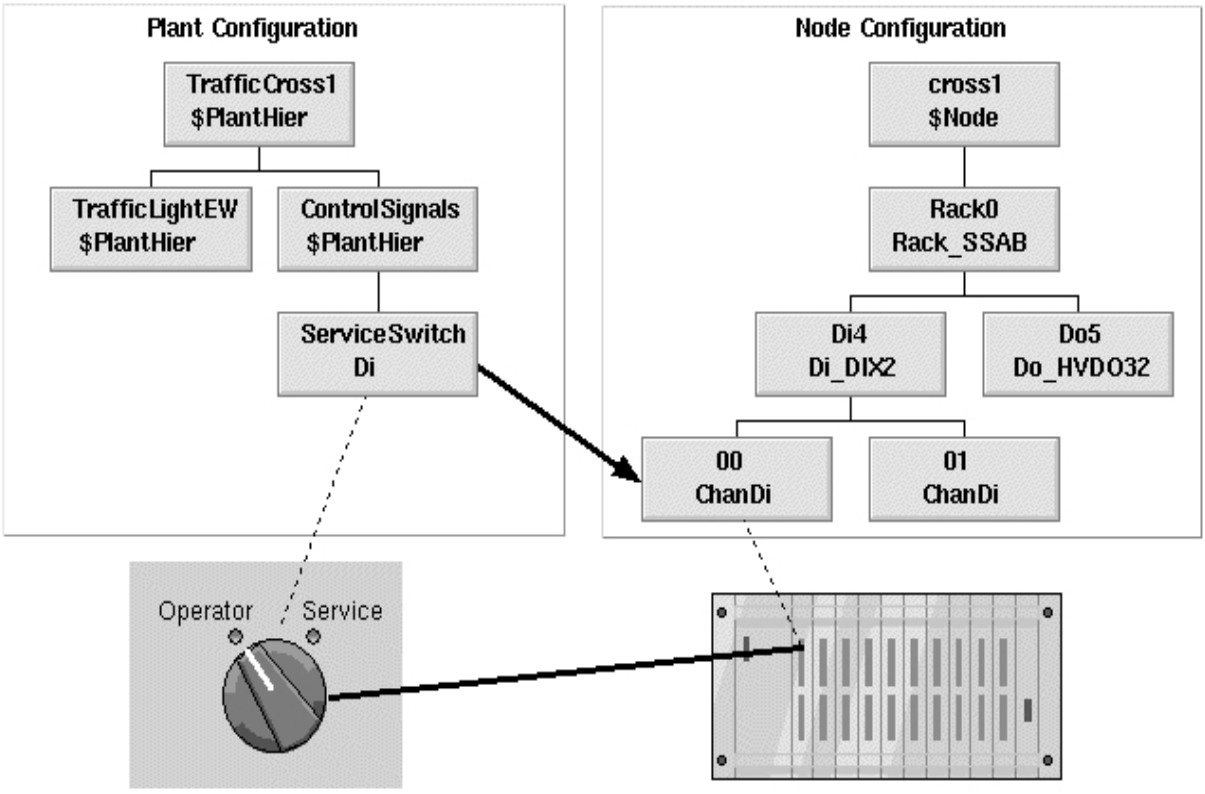


Fig Koppling mellan signal och kanal

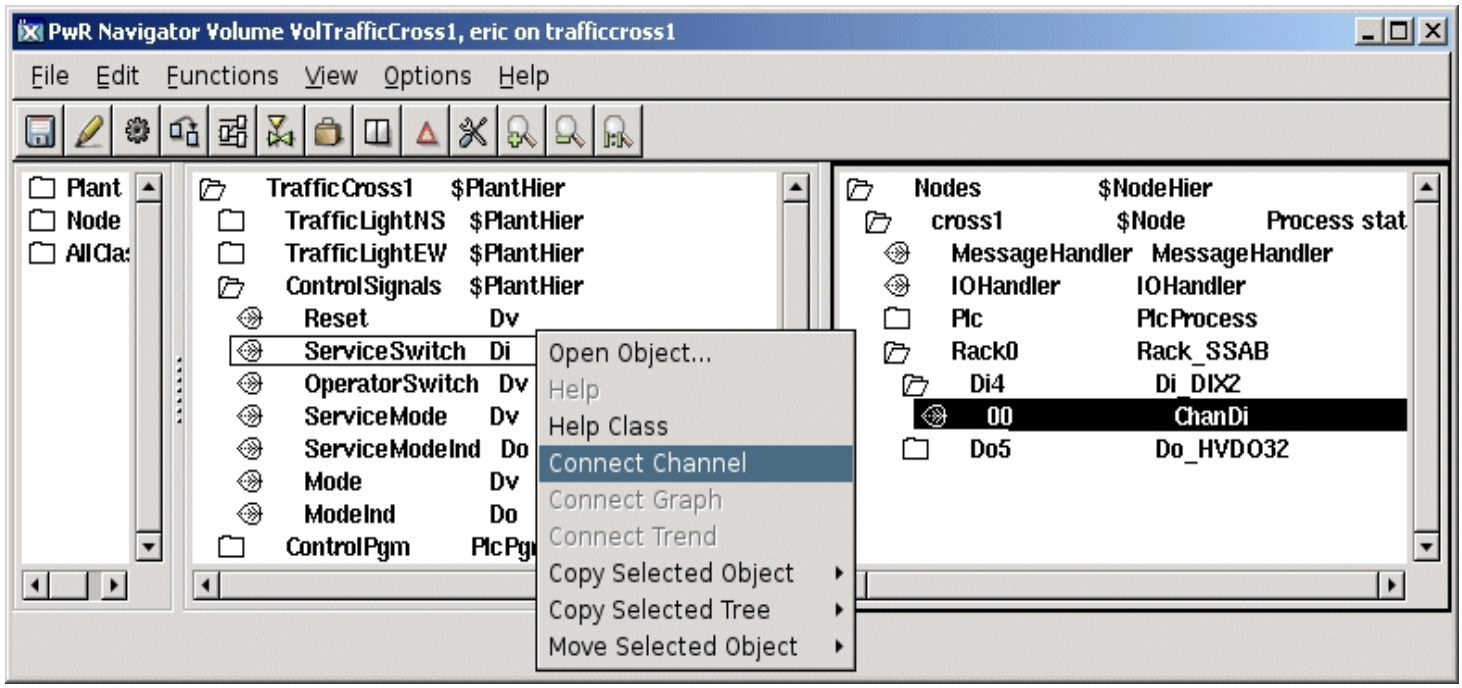


Fig Att koppla en signal till en kanal

5.5 PLC program

Vi använder den grafiska PLC editorn för att skapa PLC program.

Innan vi börjar editera kopplar vi PlcPgm objektet till ett PlcThread objekt i nodhierarkin. Detta avgör vilken tidbas som PLC programmet kommer att exekvera med.

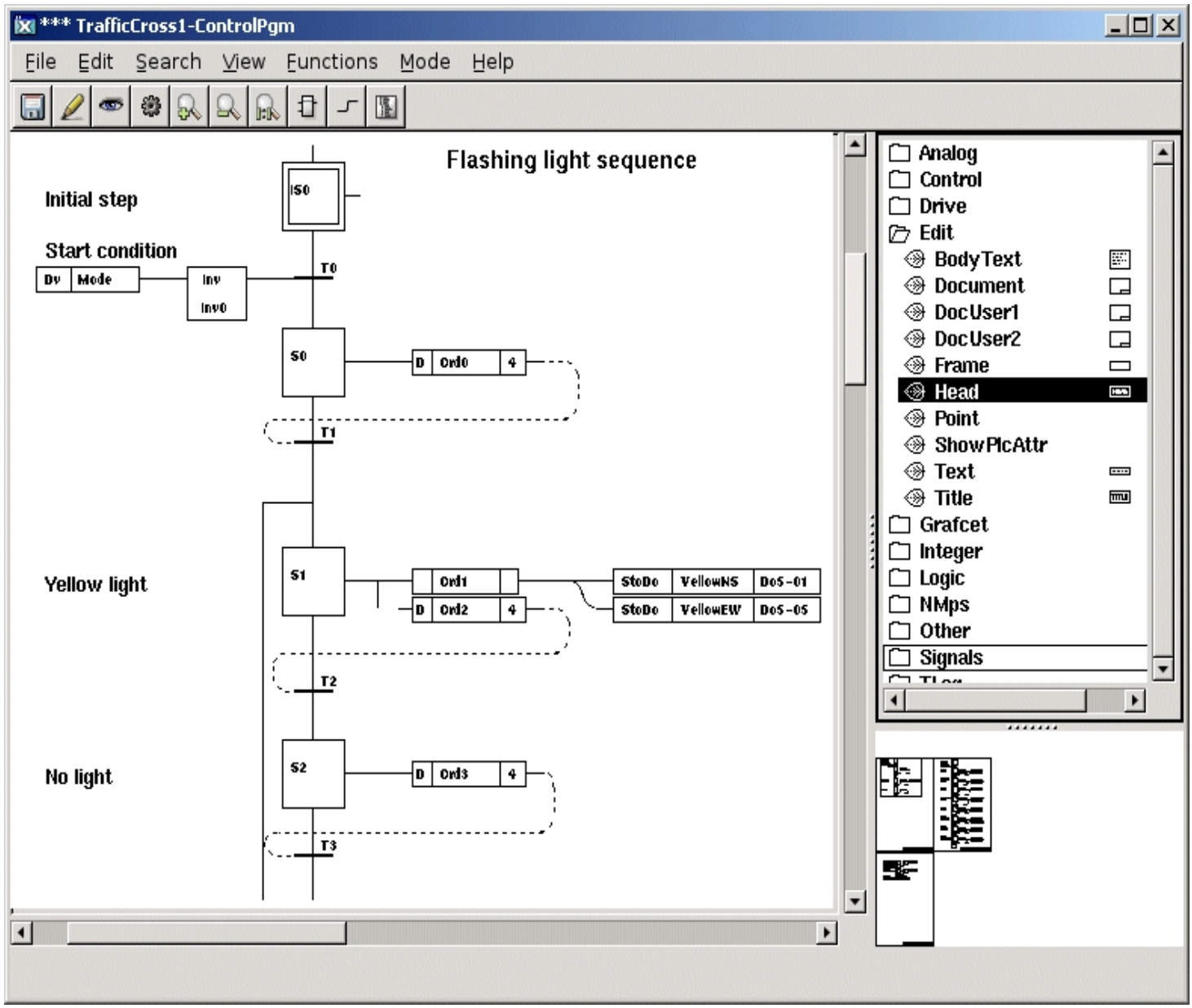


Fig Den grafiska PLC editorn

Vi kommer att använde GRAFCET för att skapa programmet för trafikljusen. Det finns två sätt att lösa problemet med de två arbets moderna normal och blinkande:

1. Använda en GRAFCET sekvens med villkorlig förgrening, en gren för den normal mod och en och en för blinkande mod.
2. Använda två separata GRAFCET sekvenser med olika startvillkor.

Här väljer vi alternativ två. En mer detaljerad beskrivning av GRAFCET och sekvensprogrammering finns i kapitel Grafisk PLC programmering.

GRAFCET program bygger på att ett aktivt tillstånd förflyttas mellan olika steg en sekvens. I linjära sekvenser kan endast ett steg vara aktivt i taget. Till vare steg knyter man ett antal order som ska exekveras när steget är aktivt. Detta kan vara att t ex sätta en digital utgångssignal, som tänder en lampa. På så sätt styr PLC programmet de logiska signaler som är kopplade till de fysiska kanalerna.

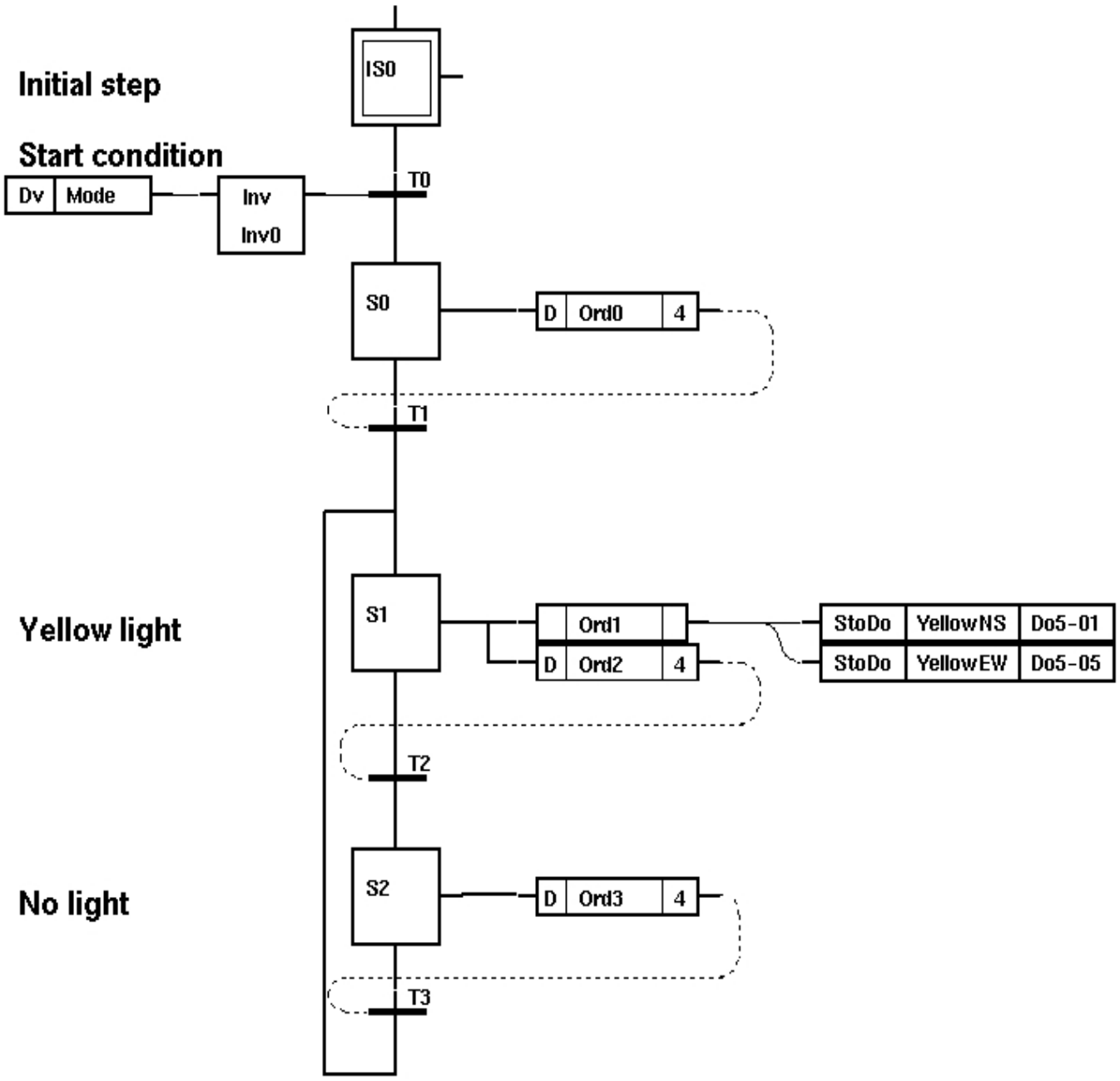


Fig Sekvens för blinkande ljus

Figuren ovan visar sekvensen som exekveras när ljusen ska blinka gult.

Startvillkoret för sekvensen är inverterat jämfört med startvillkoret för sekvensen för normal arbetsmod. Det medför att de två sekvenserna inte kommera att exekvera samtidigt.

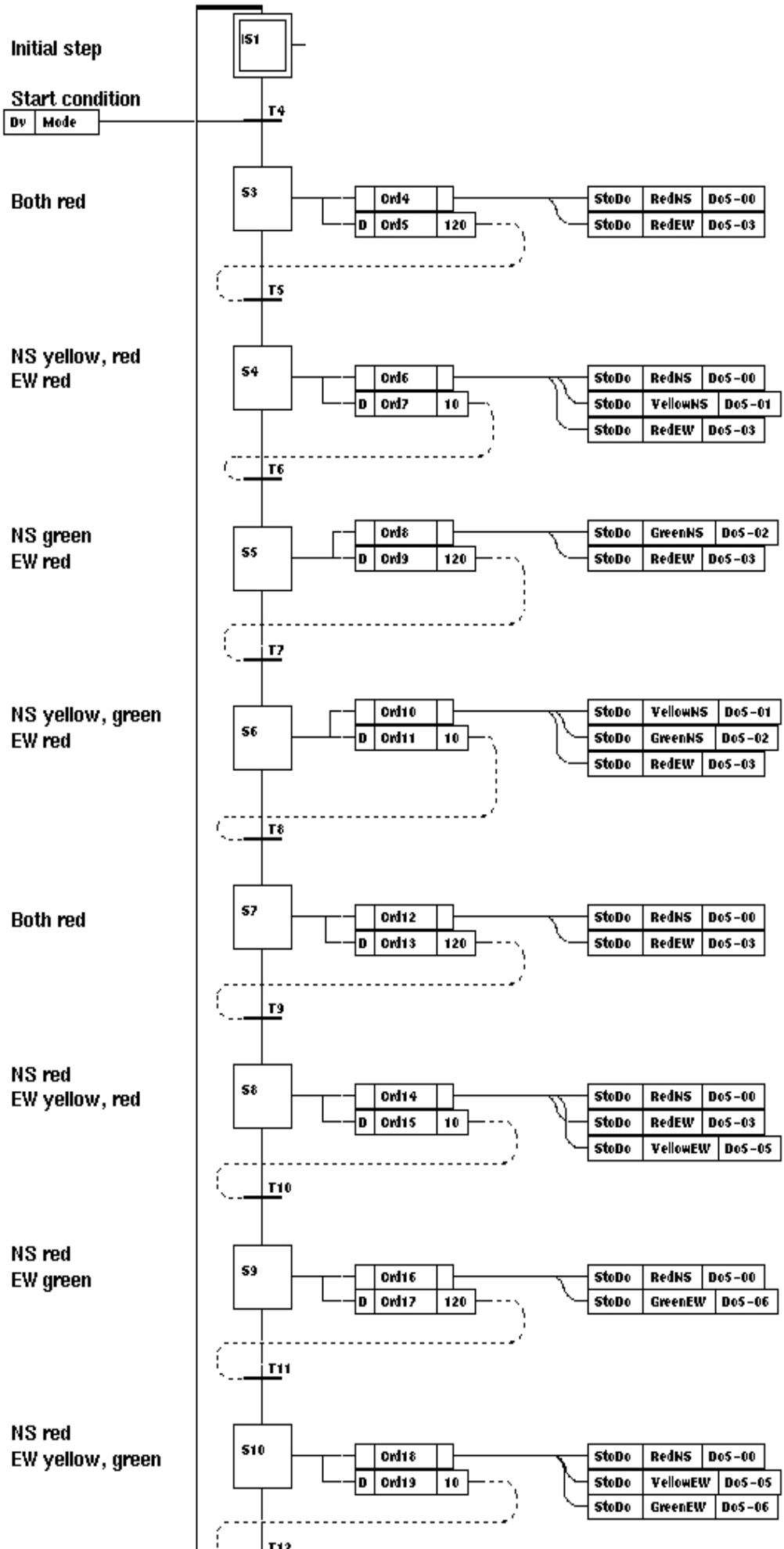

Initial step
S1
Start condition
Dv
Mode
T4
Both red
S3
Ord4
D
Ord5
120
StoDo
RedNS
Do5-00
StoDo
RedEW
Do5-03
T5
NS yellow, red
EW red
S4
Ord6
D
Ord7
10
StoDo
RedNS
Do5-00
StoDo
YellowNS
Do5-01
StoDo
RedEW
Do5-03
T6
NS green
EW red
S5
Ord8
D
Ord9
120
StoDo
GreenNS
Do5-02
StoDo
RedEW
Do5-03
T7
NS yellow, green
EW red
S6
Ord10
D
Ord11
10
StoDo
YellowNS
Do5-01
StoDo
GreenNS
Do5-02
StoDo
RedEW
Do5-03
T8
Both red
S7
Ord12
D
Ord13
120
StoDo
RedNS
Do5-00
StoDo
RedEW
Do5-03
T9
NS red
EW yellow, red
S8
Ord14
D
Ord15
10
StoDo
RedNS
Do5-00
StoDo
RedEW
Do5-03
StoDo
YellowEW
Do5-05
T10
NS red
EW green
S9
Ord16
D
Ord17
120
StoDo
RedNS
Do5-00
StoDo
GreenEW
Do5-06
T11
NS red
EW yellow, green
S10
Ord18
D
Ord19
10
StoDo
RedNS
Do5-00
StoDo
YellowEW
Do5-05
StoDo
GreenEW
Do5-06
T12

Fig Normal sekvens

Programmet för den normala arbetsmoden bygger på att trafikljusen följer sekvensen:

	Nord-Syd	**Väst-Öst**
1	Röd	Röd
2	Röd, Gul	Röd
3	Grön	Röd
4	Gul, Grön	Röd
5	Röd	Röd
6	Röd	Röd, Gul
7	Röd	Grön
8	Röd	Gul, Grön
9	Tillbaka till steg 1	

Programmet startar i initsteget. Om startvillkoret är uppfyllt, blir steg 1 aktivt och de röda lamporna tänds. Efter en viss tid blir steg 1 inaktivt, och steg 2 aktivt, och en gul lampa tänds, osv. När steg 8 har varit aktivt en viss tid, flyttas aktiviteten till initsteget igen och sekvensen börjar om, så längs startvillkoret är uppfyllt.

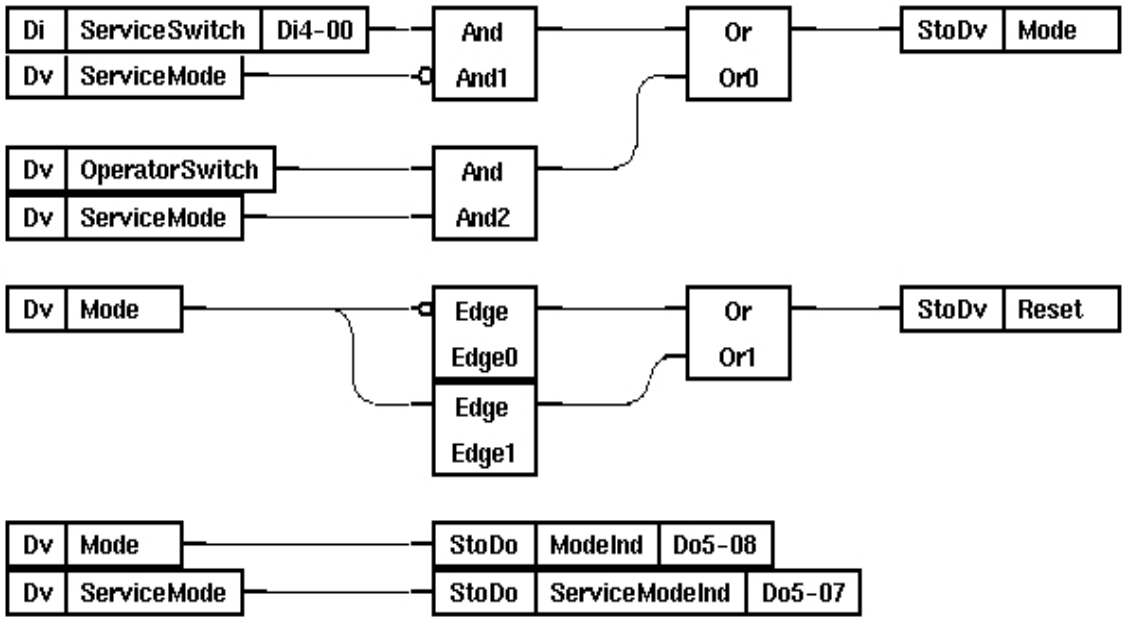


Fig Triggsignaler

Programmet ovan visar logiken för att styra olika arbetsmoder.

Längst uppe till höger sätter man Dv signalen "Mode". Om den sätt till 1 kommer sekvensen för normal arbetsmod att köras, annars körs sekvensen för blinkande ljus.

Dv signalen "Reset" kommer att vara att till 1 under ett exekverings varv när signalen "Mode" ändrar värde. Detta medför att de två GRAFCET sekvenserna återgår till initialtillståndet med initstegen aktiva. Den valda sekvensen kommer sedan att startas igen när Reset återställs till 0.

PLC programmet måste kompileras innan det kan köras på en processtation.

5.6 Processbilder

Processbilder används ofta som gränssnitt mellan operatören och processen. Processbilder skapas i den grafiska editorn.

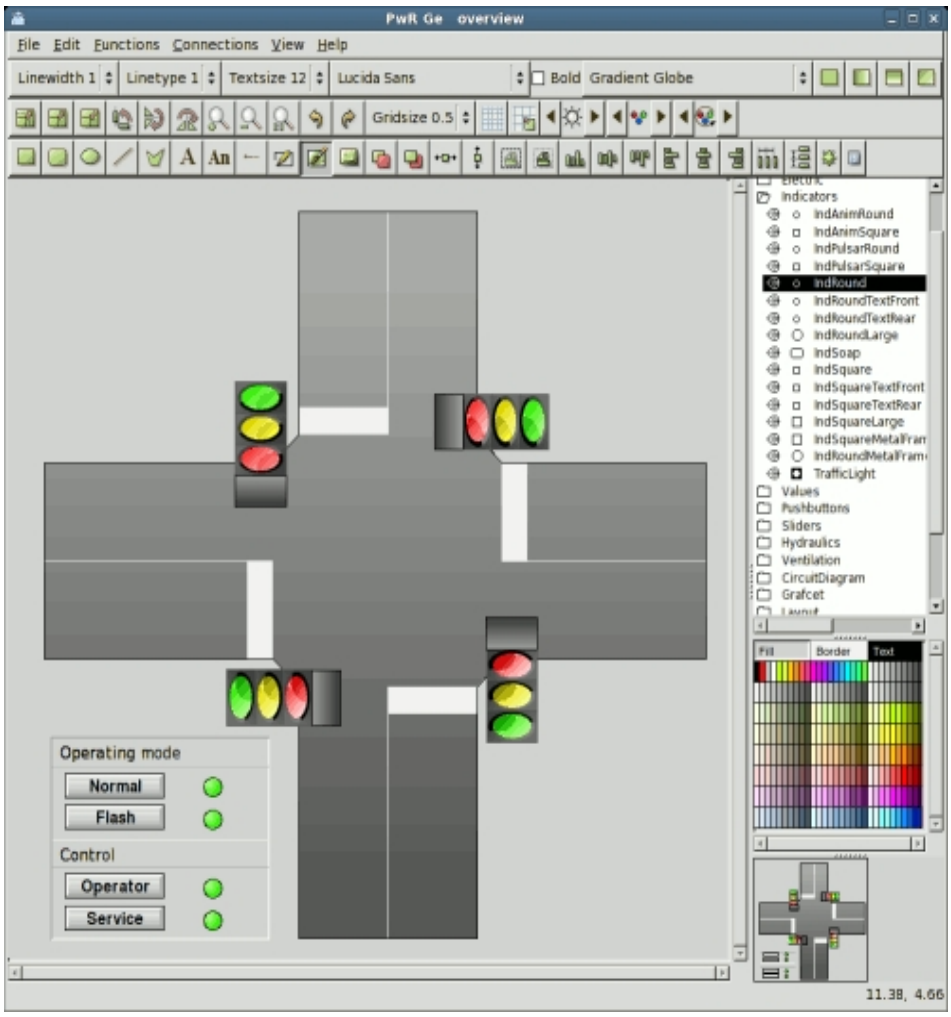


Fig Grafisk editor

Processbilder kan innehålla dynamik, som kopplas till de logiska signalerna, t ex

- Text som blir synlig när en signal har ett visst värde.
- Grafiska objekt som ändrar färg eller form när en signal för ett visst värde.
- Grafiska objekt som rör sig beroende på värdet av en signal.

Man kan även placera tryckknappar i en processbild, som operatören kan använda för att ändra värden på digitala signaler. För att ändra analoga signaler använder man inmatingsfält.

I vårt exempel väljer vi att rita en processbild som visar vägkorsningen, där trafikljusen (röda, gula och gröna) är dynamiska. Hur man skapar processbilder är beskrivet i kapitlet Skapa processbilder.

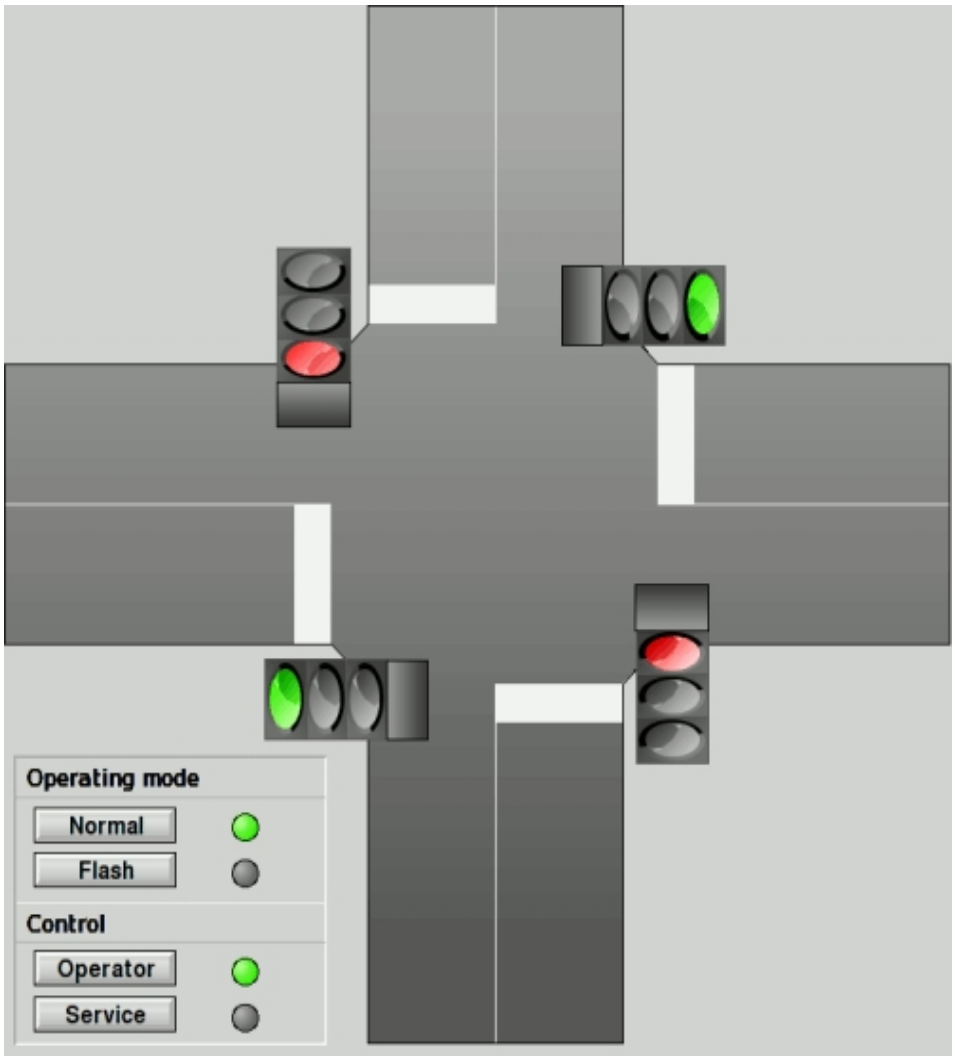


Fig Processbild för korsningen

6 Skapa ett projekt

Installation av utvecklingsmiljön

Innan man kan börja arbeta med ProviewR måste ProviewR's utvecklingsmiljö installeras. Det finns ett antal olika paket för olika Linux distributioner, och finns det inte något paket för den önskade distributionen kan man även ladda hem ProviewR källkod och bygga från denna.

Installationspaketet för utvecklingsmiljön har namngivits med versionsnummer i namnet, t ex pwr47. Detta gör det möjligt att ha flera versioner installerade parallellt, vilket är en fördel när man har många projekt i produktion som går på olika versioner.

Mer information om installation finns på Download sidan på www.proview.se.

setup script

Installationen skapar Linux-användaren 'pwrp' med password 'pwrp'. Genom att logga in som pwrp kan man starta ProviewR genom att klicka på ProviewR ikonen på skärmen.

Vill man köra ProviewR som en annan användare måste man initiera ProviewR vid inloggningen. Lägg in följande rad i filen .bashrc på hemma katalogen

```
source /src/pwrp/adm/db/pwr_setup.sh
```

Har man flera användare med gemensamma ProviewR projekt måste man se till att de har skrivrättigheter på filer i projekten. Ett sätt att uppnå det är att sätta umask till 002, dvs skrivrättighet för grupp, och ange en gemensam grupp på alla användare, t ex pwrp.

Användare

I fallstudien ovan såg vi hur man kan skapa användare som tillhör olika systemgrupper. Detta finns beskrivet i Administration kapitlet. Vid installationen av utvecklingspaketet följer det med en användardatabas med en systemgrupp, common, och fem användare, pwrp, op1, op2, op3 och op4. pwrp är utvecklings användare och har system-privilegier in runtime. op användarna är operatörer och har operatörspriviliger i runtime. De här användarna fungerar för många applikationer, och vi nöjer oss tills vidare med dessa.

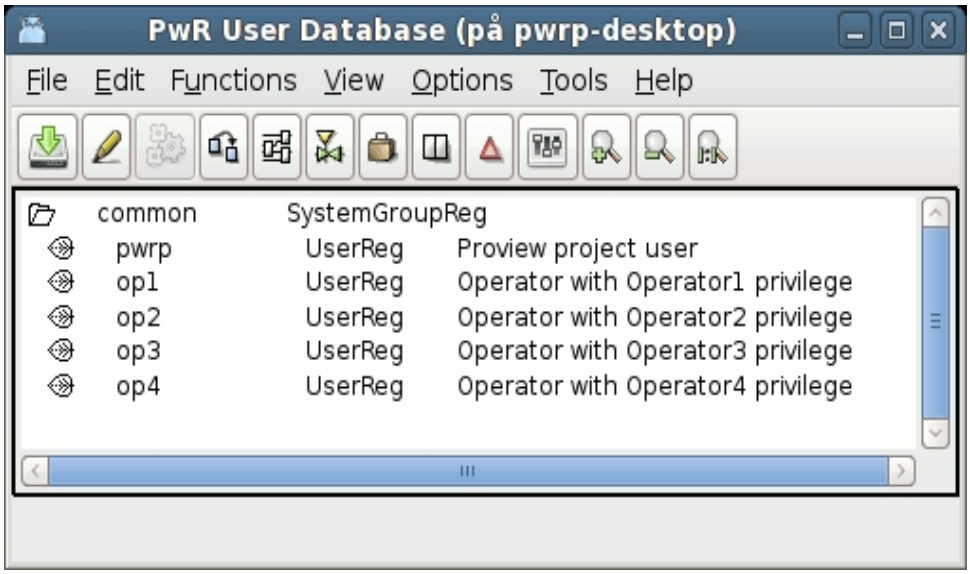


Fig Användare som följer med vid installationen

Notera att om man bygger ProviewR från källkoden, måste man skapa användare innan man kan gå vidare och skapa ett projekt.

Registrera volymer

Som vi också såg i fallstudien kan man registrera volymer i GlobalVolumeList. Man kan även välja att vänta med detta och låta konfigurerings guiden för projeket registrera volymer och hämta ut nästa lediga volymsidentitet. För att inte fördjupa oss i detaljer låter vi tills vidare konfigurerings guiden registrera volymerna. Om man har flera oberoende utvecklingsstationer bör man registrera volymerna manuellt så att volymsidentitetera inte kolliderar. Har man större anläggningar med många volymer rekommenderar vi också att man tilldelar volymsidentiteter manuellt och grupperar på lämpligt sätt.

Skapa ett projekt

För att skapa ett project går man in i administratörens projektlista, antingen genom klicka på ProviewR ikonen, eller med kommandot

```
> pwra
```

pwra är definerad som 'wb -p pwrp pwrp' dvs loggar in som användare pwrp. Har man definerat andra användare för utveckling måste man definiera om 'pwra' eller använda 'wb -p' kommandot direkt. wb har användare och password som argument.

I projektlistan går man in i editerings mod och skapar ett ProjectReg objekt, på topp-nivå eller under ett hierarki objekt. I ProjectReg objektet anger man projektnamn, ProviewR-version och katalog för projektet. När man sparar, skapas projekt-katalogerna för projektet.

Hur man skapar ett projekt finns närmare beskrivet i Getting Started Guide. Vi rekommenderar att går igenom avsnitten för att skapa projektet och konfigurere directory volymen innan man fortsätter.

7 Konfigurera Directory volymen

Öppna ett projekt

När projektet har skapas kan man hitta det i administratörens projektträd. Man öppnar ett projekt genom att aktivera 'Open Project' i popupmenyn för ett ProjectReg objekt.

Man kan också använda 'sdf' kommandot för knyta upp sig mot ett project. Projektnamnet skickas med som argument till sdf.

```
> sdf trafficcross1
```

Directory volymen kan sedan startas med

```
> pwrs
```

pwrs är definerad som 'wb pwrp pwrp' dvs loggar in som användare pwrp. Har man definerat andra användare för utveckling måste man definiera om 'pwrs' eller använda 'wb' kommandot direkt. wb har användare och password som argument (och även volym som tredje argument).

Konfiguratören

Vi öppnar nu konfiguratören för directoryvolymen. I konfiguratören finns två fönster, i det vänstra visas volymer i projektet, och i det högra noder.

Om volymen är tom, startas en guide som hjälper till med konfigureringen. För att skapa ett enkelt projekt med en nod och en volym behöver man i princip bara klicka på 'Next' knappen.

Guiden letar efter registrerade volymer för projektet. Om det inte finns några, ger förslag på lämpliga volymsnamn och lediga volymsidentiteter, och registrerar volymerna om förslagen godkänns. Guiden skapar också alla konfigurations-objekt i directory volymen och sätter lämpliga data.

Vill man senare utöka systemet med fler noder och volymer krävs att man har lite större insikt, vi gör därför en genomgång hur man konfigurerar för hand.

Konfigurera volymer

Först ska alla rotvolymer, subvolymer och klassvolymer i projekt konfigureras. Det gör man i volymsfönstret i directoryvolymen. Vi börjar med att skapa ett RootVolumeConfig objekt som konfigurerar en rotvolym.

- gå in i editeringsmod med 'Edit/Edit mode' i menyn. Nu blir paletten synlig till vänster i fönstret, och mappar kan öppna genom att klicka på mappen eller dubbelklicka på texten.
- Öppna volymsmappen och välj 'RootVolumeConfig' klassen.
- Klicka med MB2 (mittenknappen) i volymsfönstret, dvs det övre fönstret, och objektet skapas.
- Välj ut objektet och öppna objektseditorn från menyn 'Functions/Open Object'.

- Välj ut ObjectName och aktivera 'Functions/Change value' i objektseditorns meny.
- Skriv in objektets namn. Namnet ska vara detsamma som volymsnamnet.
- Stäng objektseditorn.

På samma sätt skapar man RootVolumeConfig objekt för alla rotvolymer i projektet. För nästa objekt kan man påverka positionen på objektet. Om man klickar men MB2 på objektsnamnet på ett objekt, kommer det nya objektet att bli syskon till detta objektet. Om man klickar på löv eller map-symbolen, kommer objektet att bli barn.

Även subvolumer och klassvolymer konfigureras på motsvarande sätt med SubVolumeConfig och ClassVolumeConfig objekt.

Man kan även visa attributen för ett objekt direkt i konfiguratören:

- Tryck på Shift tangenten och klicka med MB1 på objektet för att öppna objektet.
- Välj ett attribut och aktivera 'Functions/Change value' i menyn för att ändra ett värde.

Konfigurera noder

I det vänstra fönstret ska noderna i projektet konfigureras. Man delar in noderna efter vilken QCOM bus de kommunicerar på. Skapa därför två BusConfig objekt, ett för produktions noderna och ett för simulering. Öppna objekten och lägg in BusNumber.

Som barn till BusConfig objekten läggs nu ett NodeConfig objekt för varje process- och operatörsstation. När NodeConfig objektet skapas, skapas ytterligar ett par objekt

- ett RootVolumeLoad objekt som anger vilken rotvolym som ska laddas in i den här noden vid uppstart i runtime. Sätt namnet på objektet till detsamma som namnet på rotvolymen.
- ett Distribute objekt som konfigurerar som anger vilka filer som ska kopieras från utvecklingsmiljön till process- eller operatörsstationen.

Öppna NodeConfig objektet och lägg in nodnamn, operativsystem och ipaddress.

Under BusConfig objektet för simulering lägger man lämpligtvis in utvecklingsstationen, och anger som RootVolumeLoad, den processtationsvolym som man först ska börja arbeta med. Det här gör att man kan starta volymen i runtime och testa den på utvecklingsstationen. Ange utvecklingsstationens nodnamn, operativsystem och ipaddress i NodeConfig objektet.

System objekt
Skapa även ett $System objket in nodkonfigurations fönstret. Systemobjektet har attributen SystemName och SystemGroup.

- Systemnamnet är än så länge ofta identiskt med projektnamnet.
- Systemgrupps attributet gör systemet medlem av en systemgrupp i användardatabasen. vilken definierar användare för systemet. När väl systemobjektet är skapat, måste man logga in med giltigt användarnamn och passerord i arbetsbänken.

Spara
Spara sessionen med 'File/Save'. Om konfigurationen passerar syntax-kontrollen, får man frågan om man vill skapa de konfigurerade volymerna. Svara Ok på dessa frågor och skapa volymerna.

Om volymsväljaren öppnas nu, 'File/Open' i menyn, visas alla konfigurerade volymer. Nästa steg är att konfigurera en rotvolym.

8 Konfigurera en rotvolym

En rootvolym öppnas från directory volymen genom att man högerklickar på ett VolumeConfig objekt och aktiverar 'Open Project' i popupmenyn.

Om man har gjort 'sdf' till projektet kan man även öppna med 'pwrs' kommandot genom att skicka med volymsnamnet som argument.

```
> pwrs voltrafficcross1
```

Detta startar konfiguratören för rotvolymen. Liksom för directoryvolymen är den separerad i två fönster, men nu visar det övre fönstret anläggingshierarkin och det undre nodehierarkin.

Om man öppnar en tom volym, startas en guide för att konfigurera volymen. Guiden konfiguerar nod hierarkin genom att skapa diverse objekt för server processer mm, men man kan även skapa dessa för hand, se nedan.

Konfigurering av anläggningshierarkin

I anläggningshierarkin beskrivs de olika anläggningarna som finns med i ProviewR systemet. En anläggning är en logisk beskrivning av t ex en produktions process, funkioner, utrustning som ska styras och övervakas.
Se exempel på anläggningshierarki

$PlantHier objekt

Top objekten i anläggnings hierarkin är $PlantHier objekt. Detta objekt identifierar anläggningen, eller delar av den.

$PlantHier objektet används för att gruppera objekt, och strukturera anläggningen. Objektet kan t ex användas för att gruppera signalobjekt.

Signal objekt

Signalobjekten definerar en logisk signal, eller punkt, och representerar en enhet eller ett värde någonstans i processen, till skillnad från kanalobjekten som definierar fysiska signaler. Signalobjekten är generiska, dvs de kan användas med vilket I/O-system som helst.

Det finns några klasser av signaler som inte kan kopplas till kanaler, Dv, Iv, Av och Sv (DigitalValue, IntegerValue, AnalogValue and StringValue). Dessa objekt används för att lagra digitala värden, heltalsvärden, analoga värden resp teckensträngar.

Signalens värde definieras av attributet ActualValue.

Följande signalobjekt finns tillgängliga för närvarande:

Ai	Analog ingång.
Ao	Analog utgång.
Av	Analogt värde.
Ii	Heltalsingång.

Io	Heltalsutgång.
Iv	Heltalsvärde.
Di	Digital ingång.
Do	Digital utgång.
Po	Pulsad digital utgång.
Dv	Digitalt värde.
Co	Pulsgivar ingång.
Sv	Sträng värde.

Obs! PLC programmet kan läsa signaler placerade på andra noder, men kan inte skriva i dem.

PlcPgm objekt

PlcPgm objektet definierar ett PLC program. Det är möjligt att ha flera PLC program i en anläggning. Följande attribut måste datasättas:

- ThreadObject identifierar den plctråd som programmet ska exekvera i. Den refererar ett PlcThread objekt i nodkonfigurationen.
- Om programmet innehåller en GRAFCET sekvens, måste ett återställningsobjekt anges i ResetObject. Detta är en Dv, Di eller Do återställer sekvensen till utgångsläget.

Backup objekt

Backup objektet används för att peka ut ett objekt eller attribut, som ska backas upp. Det anger även om lagringen ska utföras med den snabba eller långsamma cykeln.

MountObject

MountObject objektet monterar ett objekt i en annan volym. I attributet Object specificerar man det objekt som ska monteras.

Konfigurering av nodhierarkin

I nodhierarkin definieras noderna i ProviewR systemet.

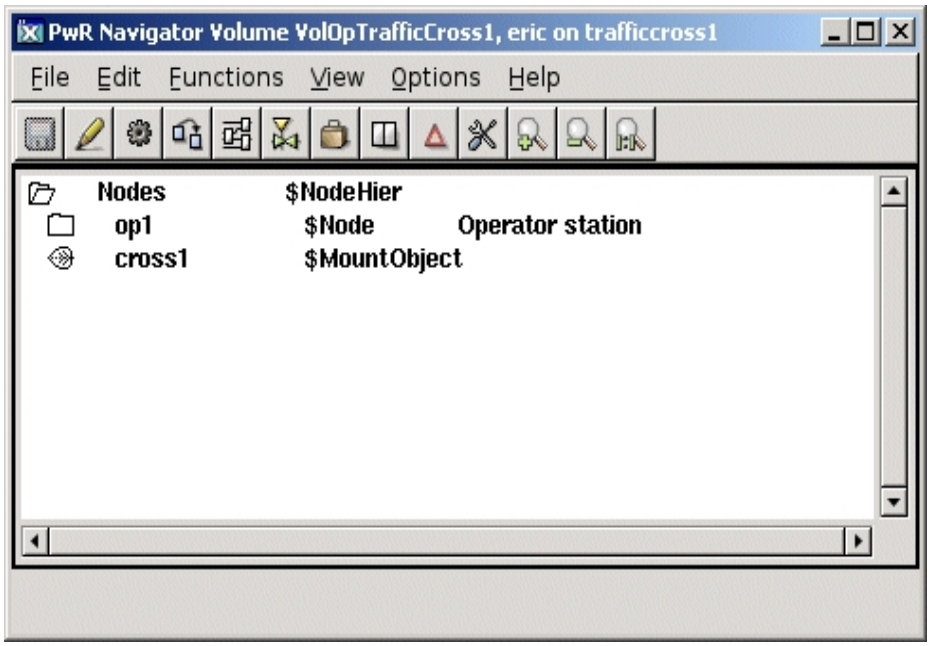


Fig Nodhierakin

$NodeHier objekt

$NodeHier objektet används för att gruppera objekt i nodhierarkin. Objektet kan användas för att gruppera t exe $Node objekt eller XttGraph objekt.
Se $NodeHier i Objektshandboken

$Node

För att definiera en nod i systemet används $Node objekt. När $Node objektet skapas, skapas dessutom ett antal server- och operatörs-objekt.
Se $Node i Objektshandboken

I/O objekt

Konfigureringen av I/O systemet är beroende på vilken typ av I/O man använder. ProviewR har ett modulariserat I/O för att kunna hantera många olika typer av I/O system: rack och kort system, distribuerade bussystem, eller system som är kopplade via olika typer av nätverk.

Det modulariserade I/O system är uppdelat på fyra nivåer: agent, rack, kort och kanal.

Rack och kort system
Vi tar I/O systemet PSS9000 som exempel på ett system med rack och kort. Systemet består av analoga och digital in- och utgångs-kort som monteras i rackar. Rackarna kopplas via en buss kabel till ett busskonverteringskort i datorn som konverterar PSS9000 bussen till datorns PCI buss.

I det här fallet används inte agent-nivån, utan $Node objektet fungerar som agent. Rack nivån konfigurerar man genom att lägga objekt av typen SSAB_Rack under $Node objektet, ett för varje rack i systemet. Korten konfigureras genom att lägga objekt specifika för olika typer av IO-kort under rack-objekten. I PSS9000 finns kortobjekt som Ai_Ai32uP, Ao_Ao4uP, Di_DIX2 och Do_HVDO32. Under kortobjekten läggs kanalobjekt, ett för varje kanal på kortet.

Gemensamt för olika I/O system är egentligen bara kanal-objekten, som definerar ut- eller ingångs-kanaler på ett kort eller en modul. Det finns några olika typer av kanaler.

ChanDi	Digital ingång.
ChanDo	Digital utgång.
ChanAi	Analog ingång.
ChanAit	Analog ingång med konvertering av signalvärde via tabell.
ChanAo	Analog utgång.
ChanIi	Heltals ingång.
ChanIo	Heltals utgång.
ChanCo	Pulsgivaringång.

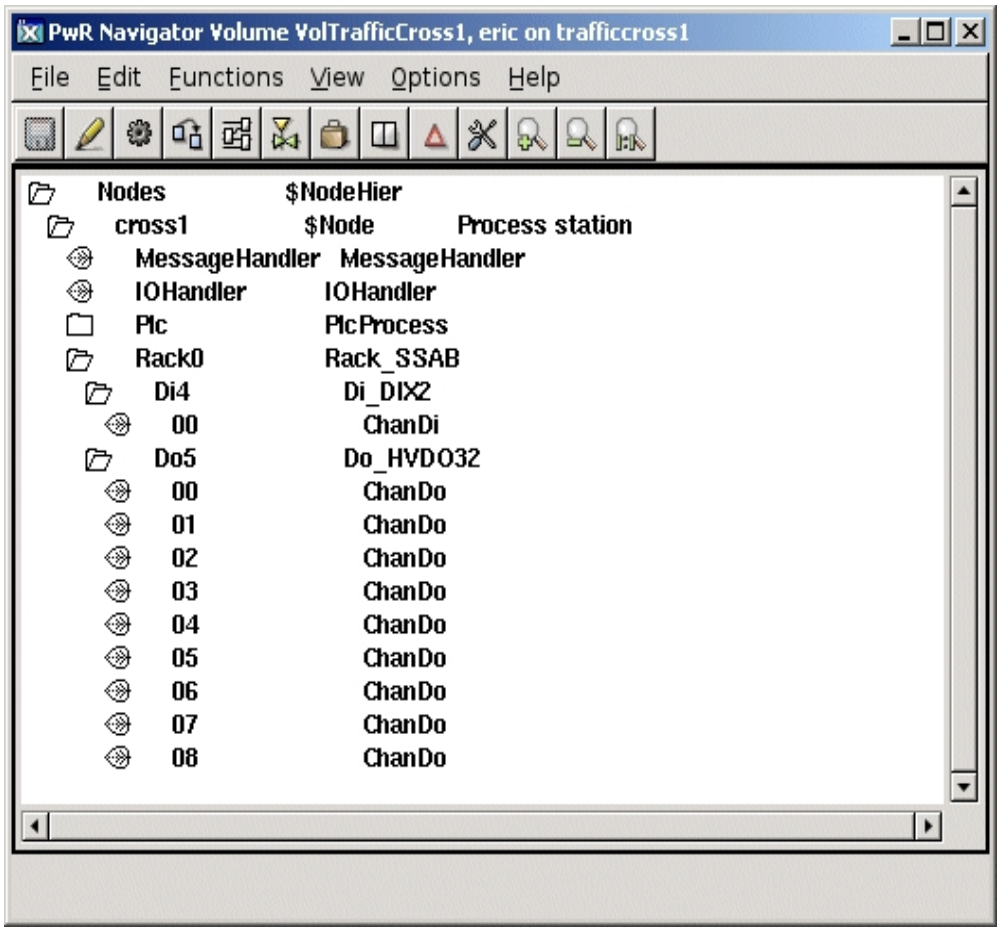


Fig I/O konfigurering

Distribuerat I/O
Som exempel på distribuerat I/O tar vi profibus. Här utnyttjas alla fyra nivåerna. På datorns PCI buss sitter ett masterkort som kommunicerar med ett antal slavar på profibusslingan. Masterkortet konfigureras med ett Pb_Profiboard objekt som ligger på agent-nivå. Under detta konfigureras de olika slavarna med Pb_DP_Slave objekt. Dessa ligger på rack-nivå. Under slav-objekten finns modul objekt av typen Pb_Ai, Pb_Ao, Pb_Di, Pb_Do etc, som ligger på kort-nivå. Under modul-objekten slutligen, konfigureras kanalerna med kanalobjekten ChanDi, ChanDo etc.

Process och tråd för I/O objekt
I/O objekt på kort-nivå innehåller ofta attributen Process och ThreadObject. I Process specificeras vilken process som ska hantera kortet.

Kortet kan hanteras av PLC programmet, vilket innebär att läsning och skrivning gör synkront med exekveringen av PLC't. Här kan man även ange vilken tråd i PLC't som ska hantera korten, dvs med vilken tidbas de ska läsas och skrivas (PlcThread attributet).

Kortet kan även hanteras av rt_io processen, som normalt går på en lägre prioritet än PLC't som ej hanterar korten synkron med PLC't. Vissa typer av analoga ingångskort som tar relativt lång tid att läsa av, hanteras med fördel av denna process.

Man kan också skriva en egen applikation som hanterar läsning och skrivning av kort. Det finns ett API för att initiera, läsa och skriva på korten. Detta kan vara av intresse om det är av vikt att läsning och skrivning av ett kort gör synkront med applikationen.

MessageHandler objekt

MessageHandler objektet konfigurerar server processen rt_emon, som hanterar övervakningsobjekten (DSup, ASup, CycleSup). När en händelse har detekterats av servern, skickas ett meddelande till de olika utenheter som är intresserade av just det här larmet.

I objektet anges t ex hur många händelser som ska lagras i noden. Objektet skapas automatisk under ett $Node objekt.
Se MessageHandler i Objektshandboken

IOHandler objekt

IOHandler konfigurerar vissa data för I/O hanteringen.

- ReadWriteFlag specificerar om man ska addressera den fysiska hårdvaran eller inte.
- IOSimulFlag anger att man simulerar den fysiska hårdvaran.
- Tidbasen för rt_io processen, dvs den process som hanterar lite långsammare typer av I/O kort som inte är lämpliga att hantera i PLC programmet.

I produktionssystem för en processtation sätts ReadWriteFlag till 1 och IOSimulFlag till 0. Om man vill simulera processtationen, t ex på utvecklingsnoden, ska ReadWriteFlag sättas till 0 och IOSimulFlag till 1.

IOHandler objektet skapas automatiskt samtidigt som ett $Node objekt.
Se IOHandler i Objektshandboken

Backup_Conf objekt

Ibland är det önskvärt att ha backup på ett antal objekt in systemet. Då placeras ett konfigurerings objekt för backupen, Backup_Conf, under nodobjektet. Backupen utförs med två olika cykler, en snabb och en långsam.

För att ange vilka objekt eller attribut som ska backas upp, använder man Backup objekt.
Se beskrivning av Backup objekt
Se Backup_Conf i Objektshandboken

Operatörsplats objekt

För att definiera en operatörsplats skapas ett objekt av klassen OpPlace under $Node objektet.

Föjande attribut måste ges ett värde:

- UserName talar om vilken ProviewR användare som ska loggas in. När operatörsmiljön startas hämtas information om användaren från ProviewR's användardatabas, bl a vilka rättigheter operatören har att påverka systemet från processbilder.
- MaxNumberOfEvents anger antalet händelser som operatörens händelselista kan innehålla samtidigt.
- EventSelectList anger vilka hierakier i anläggninghierakin, som operatören vill ta emot larm och händelser ifrån.

Om vi betraktar figuren nedan, som illustrerar anläggningen A, och antar att vi ta emot händelser från de inringade objekten anger vi 'A-C' som ett alternativ i EventSelectList. Det innebär att kommer att ta emot händelser från det övervakade objektet C, och från alla övervakade objekt under C.

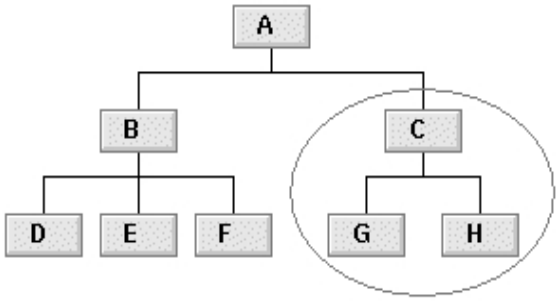


Fig EventSelectList exempel 1

Ett annat exempel:
Vi betraktar figuren nedan, som visar anläggningen TrafficCross1. Om vi vill ta emot alla händelser från TrafficCross1, lägger vi in 'TrafficCross1' i EventSelectList. TrafficCross1 hanterar två trafikljus, TrafficLightNS och TrafficLightWE. Låt oss säga att vi enbart vill ha händelser från TrafficLightNS. I det fallet anger vi 'TrafficCross1-TrafficLightNS' istället för TrafficCross1.

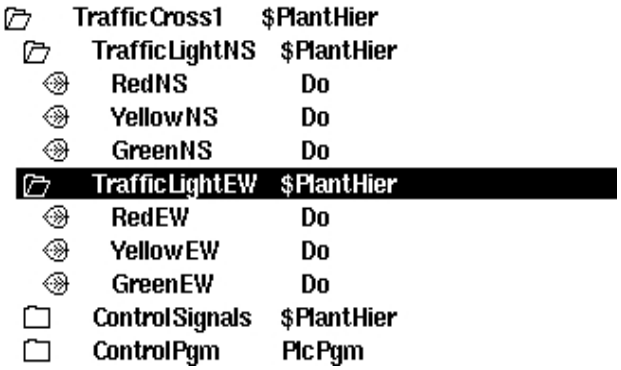


Fig EventSelectList exempel 2

Om man vill ta emot händelser från CycleSup objekt som övervakar plctrådarna, måste man även ange namnet på $Node-objektet i EventSelectList.

I FastAvail specificerar man XttGraph objekt som ska kunna startas från tryckknappar i 'Operator window'. NoFastAvail specificerar antalet tryckknappar som används. Man kan ha upp till 15 tryckknappar. Oanvända tryckknappar visas ej i Operator Window.

Se OpPlace i Objektshandboken

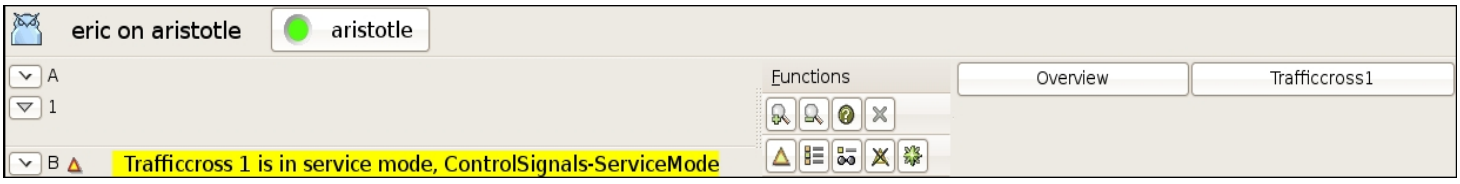


Fig Operatörsfönstret

XttGraph objekt

XttGraph objekt används främst för att visa processbilder. I objektet pekar man ut den graf som ska visas. Objekten refereras i OpPlace objektets FastAvail attribut, och i attributet DefGraph som bl a finns i $PlantHier och signal objekt och som gör det möjligt att öppna grafen från objektets popupmeny.

När objektet refereras i FastAvail kan man även utnyttja möjligheten att exekvera

xtt-kommandon mha XttGraph objektet. Man kan t ex via ett kommando sätta en viss signal från en tryckknapp i operatörsfönstret.

I XttGraph objektet ska följande attribut fyllas i:

- Action. Här anges en Ge graph, eller ett Xtt kommando.

Se XttGraph i Objektshandboken

Multiview objekt

Multiview är ett operatörs-fönster som är organiserat som en tabell, där varje cell kan innehålla en graf, trend, historisk kurva, larmlista, händelselista eller ett annat multiview fönster. Figuren nedan visar ett multiview fönster med en kolumn och två rader. The första cellen innehåller ett annat multiview fönster med två kolumner och tree rader, där den andra cellen innehåller en larmlista. De olika larmlistorna visar larm från olika delar av anläggningen, specificerade av AlarmView objekt. En multiview konfigureras med ett XttMultiView objekt. Action-vektorn innehåller specifikationerna för varje cell.

Det är också möjlig att byta ut en graf eller kurva i en cell med xtt-kommandot 'set subwindow'.

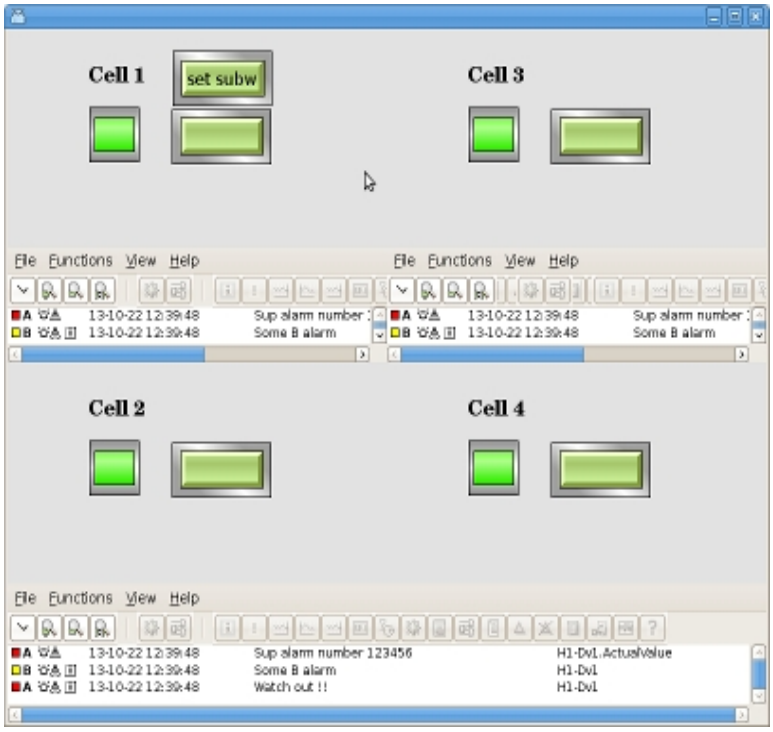


Fig Multiview window

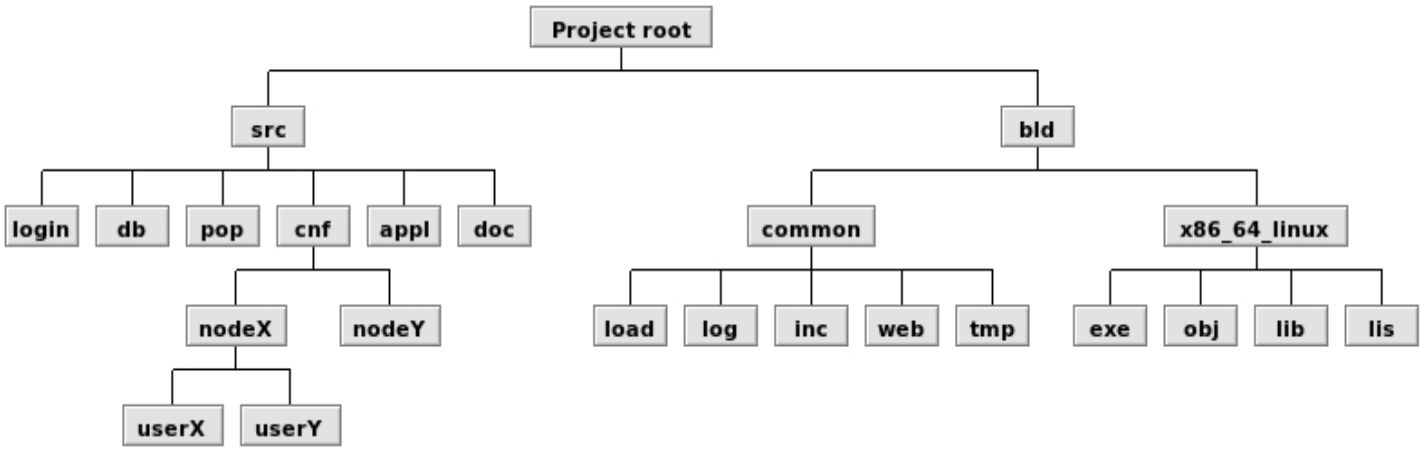
9 Navigera i projektet

Det här kapitlet beskriver projektstrukturen och förklarar avsikten med olika filkataloger. Kapitlet beskriver även en del filer som kan användas för att konfigurera olika funktioner i projektet.

Filkatalogstrukturen är uppdelad i två grenar, en som innehåller alla källkodsfiler som projektet består av, och en som är byggträdet där alla filer läggs när projektet byggs.

9.1 Introduktion

Figuren nedan visar filkatalogstrukturen för ett projekt.



Katalogstrukturen ändrades till ProviewR V4.7. Orsaken till detta var att mer särskilja källkods/konfigureringsfiler och gerenerade filer i projektet. Idén är att att allt som finns i byggträdet ska kunna återskapas från källkodsträdet.

Alla filkataloger har en omgivningsvariabel definerad för att men enkelt ska kunna nå kataloger inom projektet. Variablerna är definierade med

```
$pwrp_<directory>
```

Till exempel $pwrp_exe för katalogen <project_root>/bld/x86_64_linux/exe.

9.2 Källkodsträdet

Källkodsträdet innehåller alla filer som utgör projektets källkod och konfigureringsfiler. Topnivån $pwrp_src innehåller enbart underkataloger och inga källkods eller konfigureringsfiler. Nedan beskrivs innehållet och funktionen för underkatalogerna.

9.2.1 $pwrp_login

Den här katalogen är startpunkten när man förflyttar sig till ett projekt med 'sdf'-kommandot. Här finns två filer av intresse.

```
login.sh
sysinfo.txt
```

login.sh är ett script som körs när man går till projektet. Det kan användas för att sätta upp projekt-specifika omgivningsvariabler och liknande.

sysinfo.txt är en text-fil som kommer att skrivas ut i terminal-fönstret när man kommer till projektet. Här kan man upplysa om vad som pågår eller är gjort i projektet.

9.2.2 $pwrp_db

Det här är katalogen där databasen med alla lokala volymer ligger (inklusive direktory-volymen där projektet är konfigurerat). Varje databas ligger i egen underkatalog. Det här gäller om man väljer att skapa databaserna som BerkeleyDB databaser. Om man istället väljer att använda mysql-databaser, skapas databaserna på mysql servern.

I den här katalogen ligger också filerna för användar-definierade klasser. De är textfiler av typen .wb_load. En användarklassvolym har vanligen ett namn liknande den rotvolym som använder klasserna. Om rotvolymer heter VolMyProject kallas klassvolymen CVolMyProject (notera att detta är konvention och inte bindande). Namnet på filen blir:

```
cvolmyproject.wb_load
```

9.2.3 $pwrp_pop

Det här är katalogen där processbilderna skapade med ge-editorn sparas (filtyp .pwg). Färdiga filer kopieras till exe-katalogen i byggträdet ($pwrp_exe).

Bilder som har ett motsvarande XttGraph objekt i nodhierarkin i rotvolymen, kommer automatiskt att bli kopierade till $pwrp_exe katalogen när volymen byggs.

9.2.4 $pwrp_appl

I den här katalogen lägger man källkod for projektets applikationer, och även kod som ska länkas med plc-programmet.

En fil av speciellt intressen som ska finnas här är

```
ra_plc_user.h
```

Den här filen kommer att inkluderas när plc-koden kompileras. Den är dock inkluderad från en katalog i byggträdet (<project_root>/bld/common/inc).

Observera att ra_plc_user.h automatiskt genereras på $pwrp_inc första gången ett PlcPgm kompileras. Om du behöver modifera ra_plc_user.h bör du först kopiera filen till $pwrp_appl och lagra orginalet där.

Alla header filer som ligger här eller på underkataloger, och som ska inkluderas av plc-programmet måste kopieras till $pwrp_inc katalogen (<project_root>/bld/common/inc).

Om man har någon mindre funktion som man enbart länkar med plc-programmet och inte används någon annanstans, placerar men koden i filen

```
ra_plc_user.c
```

9.2.5 $pwrp_doc

Det här är platsen att lägga dokumentation som hör till projektet. Det kan vara egenhändigt producerad dokumentation eller till exempel datablad för komponenter i anläggningen som kanske ska distribueras till operatörsstationerna.

9.2.6 $pwrp_cnf

Katalogen innehåller alla konfigurationsfiler i projektet. Vissa konfigureringsfiler är gemensamma för hela projektet, och de är placerade här. Vissa är specifika för varje nod i projektet. Man skapar då en underkatalog för varje nod som har specifika konfigurations-filer. Ibland är en fil unik för en speciell användare. I detta fall kan man även skapa en underkatalog i nod-katalogen för den användaren.

Filer som läggs här är:

Konfigurering av globala funktionstangenter.
`Rt_xtt`

Konfigurering av menuer och kortkommandon i rt_xtt.
`xtt_setup.rtt_com`

Startupfil för ProviewR.
`ld_appl_<nodename>_<bus_no>.txt`

Fil för att ange bibliotek som länkas med plc-programmet.
`plc_<nodename>_<bus_no>_<plc_name>.opt`

Fil för att sätta initialvärden vid ProviewR uppstart.
`pwrp_alias.dat`

Fil med hjälptexter
`xtt_help.dat`

Alla ovanstående filer beskrivs närmare längre fram.

9.3 Byggträdet

Byggträdet innehåller alla filer som produceras när man bygger volymer och noder. Källkoden ligger däremot i källkodsträdet. Idén är att byggträdet i princip ska kunna tas bort och återskapas från källkodsträdet.

9.3.1 $pwrp_exe

Detta är katalogen där exe-filer för plc program skapas. Plc-program namnges:

`plc_<nodename>_<bus_no>_<version_no>`

Om man har applikationsprogram i projektet ska även exe-filerna för dem ligga här.

9.3.2 $pwrp_obj

På den här katalogen läggs alla objektsfiler som skapas vid kompilering. Objektsfiler för plc-programmen läggs här automatiskt. En objektsfil för ett PlcPgm objekt i anläggningshierarkin namnges efter objektets identitet.

Objektsfiler för all annan kod ska också placeras här.

9.3.3 $pwrp_lis

Här placeras list-filer som produceras vid kompilering.

9.3.4 $pwrp_lib

På den här katalogen skapas bibliotek som innehåller objektsfiler för en viss volym. Man ska även placera andra bibliotek här.

9.3.5 $pwrp_load

Det här är katalogen där laddata-filer för projektets volymer skapas. Laddata-filen namnges:

```
<volumename>.dbs
```

9.3.6 $pwrp_log

Den här katalogen innehåller log-filer som skapas vid simulering av projektet. ProviewR's systemlogg-fil heter

```
pwr_<nodename>.log
```

Om man startar om simuleringen, kommer loggningen att adderas till filen. Ta bort filen om gammal loggningen ska rensas bort.

9.3.7 $pwrp_tmp

Katalog för temporär-filer. De här filerna skapas vid vissa operationer, till exempel när ett plc-program kompileras i debug-mod. Då lagras källkods-filerna här för att sedan kunna användas av debuggern.

9.3.8 $pwrp_inc

Det här katalogen innehåller include-filer som inkluderas vid bygge av plc-program.
En fil kallad
`ra_plc_user.h`
letas alltid efter i denna katalog. Om man har andra header-filer som inkluderas, ska de inkluderas i ra_plc_user.h. Källkoden till include-filerna ska ligga i källkodsträden och kopieras därifrån till $pwrp_inc.

Header-filer för användarklasser skapas här när man bygger en klassvolym. Filerna namnges:

```
pwr_<classvoumename>classes.h
pwr_<classvoumename>classes.hpp
```

9.3.9 $pwrp_web

Katalogen innehåller alla filer för web-gränssnittet. Till exempel pwg-filer för Ge grapfer, flw-filer för plc trace coh xtt-hjälpfiler konverterade till html-filer.

9.4 Speciella filer

Här beskrivs speciella filer som används för olika typer av konfigurering, eller som är av annat intresse, och är förlagda någonstans i projekt-trädet. De är alla nämnda tidigare i det här kapitlet.

9.4.1 Rt_xtt

Den här filen läses in vid start av rt_xtt, och hämtas från katalogen varifrån rt_xtt startas. Filen konfigurerar tangenter för att utföra olika typer av kommandon.

Giltiga kommandon är:

```
Command     // Exekverar ett xtt kommando.
SetDig      // Sätter en digital signal till TRUE.
ToggleDig   // Toggla värdet på en digital signal.
ResetDig    // Återställer en digital signal till FALSE.
```

För att koppla en tangent till ett kommando måste man först definera tangenten och sedan kommandot. Ett exempel för att koppla tangenttryckningen <Ctrl>F5 till ett kommando som kvitterar A-larm:

```
Control <Key>F5: Command(event ack /prio=A)
```

En typisk Rt_xtt fil kan se ut så här:

```
#
# Function key definition file
#
Control <Key>F5: Command(event ack /prio=A)
Control <Key>F6: Command(event ack /prio=NOA) # ack non A-alarms
Control <Key>F7: Command(show alarm)          # open alarm list
Control <Key>F8: Command(show event)          # open event list
# Below opens a graph defined by a XttGraph-object in the node hierarchy.
# The $Node-expression will be replaced by the node-object on this node.
# This makes the Rt_xtt-file work on different nodes.
Alt <Key>F12: Command(open graph /object=$Node-Pictures-rkt_overview)
# Below opens a graph defined by a XttGraph-object in the node hierarchy.
# The /focus-syntax sets focus on a object in the graph named NewPlate
Control <Key>F9: Command(open graph /object=*-Pictures-rkt_platepic/focus="NewPlate")
Control <Key>F10: Command(open graph /object=*-Bilder-rkt_cells/focus="Check_no")
# Below closes all open graphs except rkt_overview.
Control <Key>F11: Command(close all/except=rkt_overview)
Shift <Key>F1: SetDig(VWX-RKT-RB-DS-OnOffMan_1_2.ActualValue)
Shift <Key>F6: ResetDig(VWX-RKT-RI-RP-CalcPrePos.ActualValue)
Shift Control <Key>v: ToggleDig(VWX-RKT-COM-VWXSVR-BlockOrder_RKT.ActualValue)
```

9.4.2 xtt_setup.rtt_com

Den här filen läses in vid start av rt_xtt. Filen läses från den katalog man startar rt_xtt från. Med hjälp av den kan man konfigurera egna menyer och koppla menyalternativ till att utföra speciella kommandon.

Alla kommandon i filen följer syntaxen för xtt script och kommandon. Se Operatörshandboken för mer info om script och kommandon.

När man startar rt_xtt finns som default en meny med alternativet Databas överst. Om man, till exempel, vill skapa en meny ovanför denna är kommandot:

```
create item/text="Maintenance"/menu/dest=DataBase/before
```

För att skapa en undermeny (första barnet) till denna används kommandot:

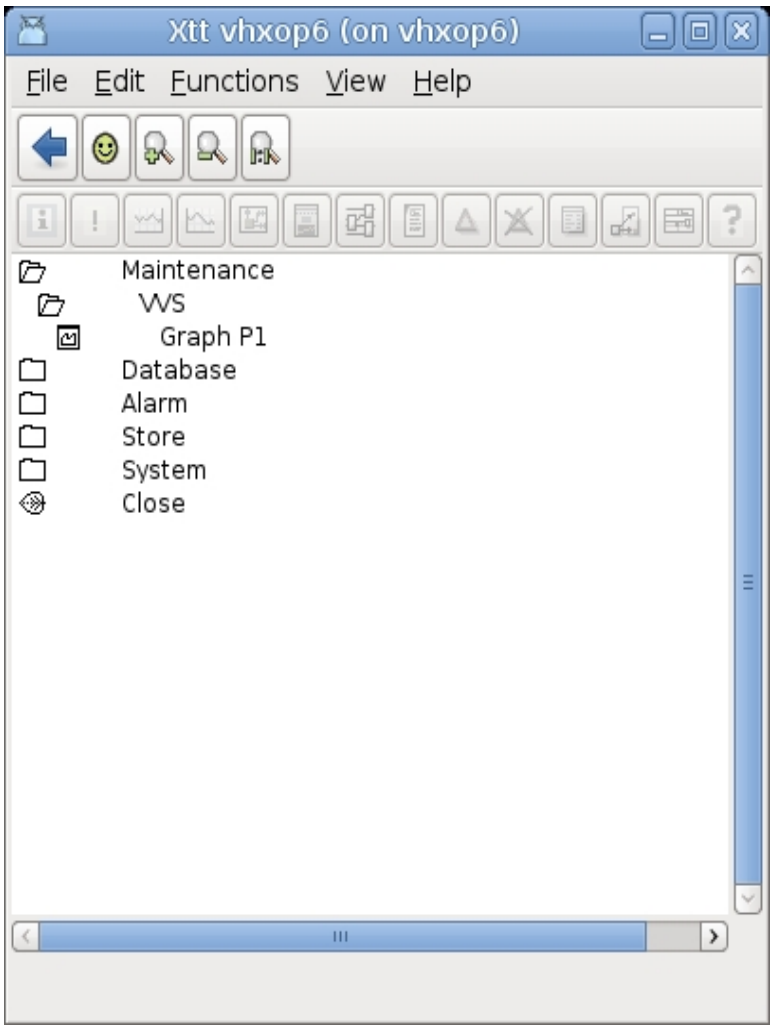
```
create item/text="VVS"/menu/dest=Maintenance/firstchild
```

För att skapa ett menyalternativ som utför ett kommando under VVS menyn används kommandot:

```
create item/text="Graph P1"/command="open graph/object=$Node-pics-h4_proc1"\
       /pixmap=graph/dest=Maintenance-VVS/lastchild
```

Pixmap kvalifieraren anger att en speciell ikon används för menyalternativet. Utan denna blir ikonen en löv. Kommandot öppnar en graf definierad med ett XttGraph objekt i nodhierarkin.

Resultatet ser ut så här:



Förutom att bygga menyer, kan man även definera symboler (genvägar) som kan användas som kommandon. Symbolen matas in på kommando-raden och kommandot definierat av symbolen exekveras.
Nedan defineras symbolen 'h4' som öppnar en graf:

```
define h4 "open graph /object=*-pics-h4_process1"
```

I xtt_setup-filen lägger man in ett kommando per rad. Kommentars-rader inleds med ett utropstecken.

9.4.3 ld_appl_<node>_<bus_no>.txt

I den här filen anges de applikationer som ska startas vid start av ProviewR runtime. Man kan addera egna appikationer, men även stänga av en eller flera av ProviewR's system-processer.

En typisk ld_appl-fil kan se ut så här:

```
# Startup file for Proview/R
#
# id,   name,   load/noload run/norun,  file,   prio,   debug/nodebug,  "arg"
#pwr_neth,      , noload, norun, , 5, debug, ""
#pwr_plc,       , noload, norun, ,  , debug, ""
#pwr_alim,      , noload, norun, , 5, debug, ""
#pwr_emon,      , noload, norun, , 5, nodebug, ""
#pwr_tmon,      , noload, norun, , 5, debug, ""
#pwr_qmon,      , noload, norun, , 19, debug, ""
#pwr_nacp,      , noload, norun, , 5, debug, ""
#pwr_bck,       , noload, norun, , 5, debug, ""
#pwr_io,         , noload, norun, , 5, debug, ""
#pwr_linksup,   , noload, norun, , 5, debug, ""
#pwr_trend,     , noload, norun, , 5, debug, ""
#pwr_fast,       , noload, norun, , 5, debug, ""
#pwr_remh,       , noload, norun, , 5, debug, ""
pwr_remlog,       , noload, norun, , 5, debug, ""
#pwr_sysmon,       , noload, norun, , 5, debug, ""
#pwr_elog,       , noload, norun, , 5, debug, ""
pwr_websocketserver,       , noload, norun, , 5, debug, ""
pwr_webmonmh,       , noload, norun, , 5, debug, ""
pwr_webmonelog,       , noload, norun, , 5, debug, ""
#pwr_opc_server,       , noload, norun, , 5, debug, ""
#pwr_sevhistmon,       , noload, norun, , 5, debug, ""
#pwr_sev_server,       , noload, norun, , 5, debug, ""
#rs_nmps_bck, rs_nmps_bck, noload, run, rs_nmps_bck, 12, nodebug, ""
ra_utl_track, ra_utl_track, noload, run, ra_utl_track, 12, nodebug, ""
```

'#' teckenet innebär att raden är bortkommenterad. Nästan alla ProviewR's systemprocesser är bortkommenterade eftersom vi vill att de ska starta (de är markerade med 'norun'). Om vi tar bort #-tecknet kommer denna systemprocess ej att startas. I ovanstående exempel har vi inte något web-gränssnitt och startar därför inte de processen som hanterar detta (pwr_websocketserver).

På den sista raden finns en applikation som tillhör projektet och ska startas vid ProviewR start. Prioriteten är satt till 12 som är något under prioriteten för ProviewR's systemprocesser som har mellan 17 och 19.

9.4.4 plc_<node>_<bus_no>_<plc_name>.opt

OBS! Den här filen är ersatt av BuildOptions objektet sen V4.8.2

Den är filen (om den existerar) används för att addera bibliotek och objektsmoduler vid länkningen av plc-programmet. Några bibliotek och objektsmoduler måste finnas med, och en default option-fil ska se ut så här:

```
$pwr_obj/rt_io_user.o -lpwr_rt -lpwr_usbio_dummy -lpwr_usb_dummy -lpwr_pnak_dummy
 -lpwr_cifx_dummy -lpwr_nodave_dummy -lpwr_epl_dummy
```

Till detta adderas de bibliotek och moduler som behövs. Syntaxen är den som gäller för ld (The GNU linker). ProviewR genererar en template option-fil .opt_template som kan användas som mall.

9.4.5 pwrp_alias.dat

Fil för att sätta initialvärden vid ProviewR start.

Det finns några olika sätt att sätta värden med pwrp_alias-filen.
Samma fil används för alla noder i projktet. Varje rad ska inledas med följande uttryck:

```
<nodename>_setval
```

De olika sätten att sätta värden beskrivs nedan:

1. Sätta värde på ett attribut

```
<nodename>_setval <attribute_name> = <value>
```

Exempel

```
bslds1_setval bsl-ds1-par-maxtemp.actualvalue = 70.0
```

Med denna syntax kommer värdet att sättas före backup'en har laddads, och före plc-programmet har startat. Det betyder att om värdet backas upp är det backupvärdet som gäller.

Om man vill vara säker på att datasättningen kommer att ha effekt används denna syntax:

```
<nodename>_setvalp <attribute_name> = <value>
```

I det har fallet utförs datasättningen efter backup-filens laddning och plcprogrammets start.

2. Sätta simulerings-mod

Att sätta simulerings-mod innebär att I/O inte hanteras. Man kan skriva simulerings-program för att sätta lämpliga värdet på ingångs I/O. Addera den här raden till filen:

```
<nodename>_setval plcsim = yes
```

3. Stäng av exekveringen av alla plc-program vid startup

För att sätta scan-off på alla plc-program används den här raden:

```
<nodename>_setval plcscan = off
```

Exekveringen på ett PlcPgm kan sedan sättas på genom att sätta ScanOff attributet i WindowPlc objektet (ligger som barn till PlcPgm-objektet) till 0. Observera att eventuella underfönster också kan behöva startas.

9.4.6 /etc/proview.cnf

En konfigurations-fil för definition av diverse parametrar både i utvecklings-, runtime- och lagrings-miljön. På en utvecklings-station gäller definitionerna alla projekt på noden.

Parametrar i utvecklings-miljön

mysqlSocket	mysql socket för volymer med mysql databas. Defaultvärde "/var/run/mysqld/mysqld.sock".
mysqlServer	Server node för mysql.
defaultProjectRoot	Default rot-katalog för projekt när nya projekt skapas. Defaultvärde "/usr/local/pwrp".
defaultProductionQBus	Default produktionsbuss när nya projekt skapas.

	Defaultvärde 1.
defaultSimulationQBus	Default simuleringsbuss när nya projekt skapas. Defaultvärde 999.
defaultSystemGroup	Default systemgrupp när nya projekt skapas. Defaultvärde "Common".
defaultNodeHierRoot	Default namn på översta objektet i node-hierarkin när nya projekt skapas. Defaultvärde "Nodes".
defaultSecurity	Default namn Security objektet. Defaultvärde "Security".
defaultXttPriv	Defaultvärde på XttPriv i Security-objektet. Defaultvärde "Security".
defaultOpPlaces	Default namn på map för OpPlace-objekt i node-hierarkin. Defaultvärde "OpPlaces".
defaultServers	Default namn på map för server-objekt i node-hierarkin. Defaultvärde "Servers".
defaultIO	Default namn på map för IO-objekt i node-hierarkin. Defaultvärde "IO".
defaultApplications	Default namn på map för applicationsobjekt i node-hierarkin. Defaultvärde "Applications".
defaultOpOp	Default namn på OpPlace-objekt för operatörer i node-hierarkin. Defaultvärde "Op".
defaultOpMaintenance	Default namn på OpPlace-objekt för underhåll i node-hierarkin. Defaultvärde "Maintenance".
defaultOpDefault	Default namn på default OpPlace-objekt i node-hierarkin. Defaultvärde "OpDefault".
defaultWebBrowser	Default namn på WebBrowser-objekt i node-hierarkin. Defaultvärde "WebBrowser".
qcomBusId	QCom bus identitet för simulering.

Parameterar i runtime-miljön

qcomBusId	QCom bus identitet.
curveExportFile	Default fil-namn vid export från en historik kurva.
webDirectory	Katalog där web-filer placeras och läses av webservern.

Parameterar i lagrings-miljön

sevDatabaseType	Databastyp, mysql eller sqlite.
sevMysqlEngine	Mysql engine för tabeller som skapas, innodb eller myisam.
sevTimeout	Timeout tid i sekunder för hämtning av historisk data från sev serv

10 Grafisk PLC programmering

Det här avsnittet beskriver hur man skapar PLC program.

Editorn

Man startar editorn från ett PlcPgm objekt i anläggningshierarkin. Välj ut objektet och aktivera 'Functions/Open Program' i menyn. Den första gången programmet öppnas, finns där ett tomt dokument objekt. Programmet består av funktionsobjekt och Grafcet sekvenser.

Programmering med funktionsblock görs i ett horisontellt nät av noder och förbindningar, från vänster till höger i dokumentet. Värden på signaler och attribut hämtas upp på den vänstra sidan av nätet, och värden överförs med förbindningar från utgångspinnar till ingångspinnar på funktionsblocken. Funktionsblocken opererar på signalvärdena och i den högra sidan av nätet lagras värdena i signaler eller attribut.

Grafcet sekvenser består av ett vertikalt nät av noder och förbindningar. Ett tillstånd överförs mellan stegen i sekvensen via förbindningarna. Nät av Grafcet och funktionsblock kan samverka med varandra och kombineras till ett nät.

Editera funktionsobjekt

PLCeditorn består av
- en arbetsarea.
- en palett med Grafcet objekt och funktionsblock, och en palett med förbindningar.
- ett navigationsfönster, från vilket arbetsarean kan skrollas och zoomas.

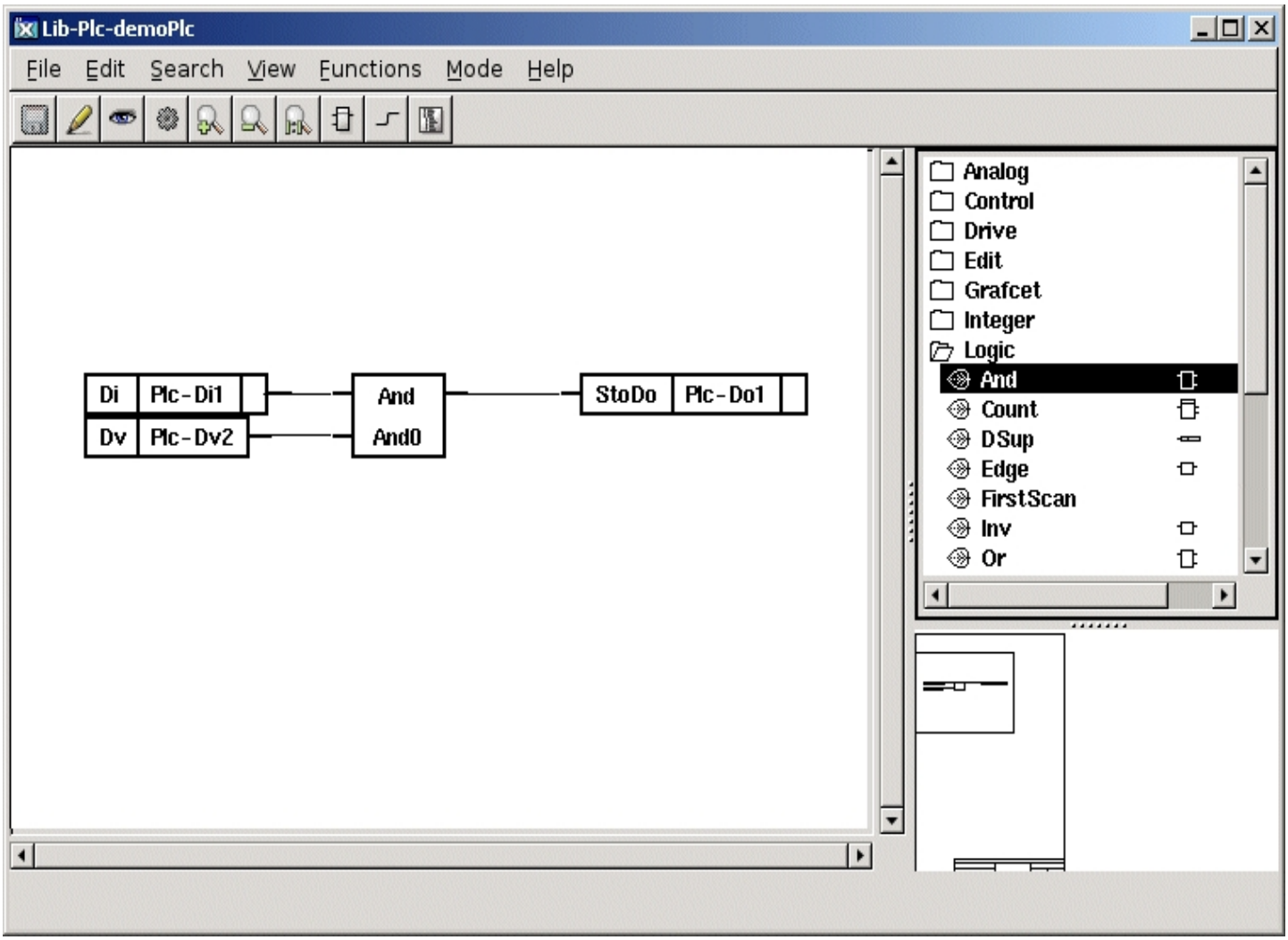


Fig Plc editorn

Ett funktionsobjekt skapas genom att man väljer en klass i paletten, och klickar med MB2 (mittenknappen) i arbetsarean.

Modifiera ett objekt

Ett objekt kan modifieras från objektseditorn. Denna öppnas genom att välja ut objektet och aktivera 'Functions/Open Object' i menyn. Värden på objektets attribut kan ändras med 'Functions/Change value' i objekteditorns meny, eller trycka på tangenten Pil Vänster. Om en in- eller ut-gångspinne inte används kan den tas bort med en checkbox. Det finns också en checkbox som avgör om en digital ingång ska vara inverterad.

Koppla ihop funktionsobjekt

En utgångspinne och en ingångspinne kopplas ihop genom att

- placera markören på pinnen, eller på ett område i funktionsobjektet nära pinnen, och tryck ned MB2.
- drag markören till den andra pinnen, eller till ett område i funktionsobjektet nära pinnen, och släpp MB2.

En förbindning skapas nu mellan funktionsobjekten.

Hämta värdet på en signal

Värdet på en Di signal hämtas med ett GetDi objekt. GetDi objektet måste peka på Di objektet och detta görs genom att välja ut signalen i konfiguratören, och sedan trycka ned Ctrl tangenten och dubbelklicka med MB1 på GetDi objektet. Signalnamnet visas nu i objektet. Dv signaler, Do signaler och attribut hämtas på samma sätt med GetDv, GetDo resp GetDp objekt.

Enklaste sättat att skapa ett Get objekt är att rita en koppling från den kopplingspunkt som Get-objektet ska vara kopplat till, och släppa den på ett tomt ställe i arbetsarean. Nu skapas ett generiskt Get-objekt som omvandlas till ett Get-objekt av rätt typ när signalen specificeras Signalen specificeras som tidigare genom att välja ut signalen i anläggnings-hierarkin och klicka med Ctrl/dubbelklick på Get-objektet.

Lagra ett värde i en signal

Värdet från en utgång på ett funktionsobjekt lagras i en Do signal med ett StoDo objekt. StoDo objektet kopplas till Do signalen på samma sätt som Get objekt. Dv signaler och attribut lagras med StoDv resp StoDp objekt.

Grafcet

Den här sektionen ger en första introduktion till Grafcet. För en mer detaljerad beskrivning, läs referensmanualen för Grafcet. Grafcet är en internationell standard för sekvensstyrningar.

Grafcet består av ett antal steg, och till varje steg är ett eller flera order kopplade, som kommer att exekveras när steget är aktivt. Från början är initsteget aktivt, och aktiviteten flyttas till nästa steg med ett övergångsvillkor. En förflyttning görs när villkoret för ett övergångsvillkoret är uppfyllt.

Enkel rak sekvens

Vi tittar på den enkla sekvensen nedan och antar att initsteget (Start) är aktivt, vilket betyder att ordern som är kopplad till initsteget (Order1) kommer att utföras. Ordern kommer att utföras ända tills initsteget blir inaktivt. Steg 1 (Step1) blir aktivt, när övergångsvillkoret T1 är uppfyllt. Initsteget blir då inaktivt.

En Grafcet sekvens är alltid en sluten sekvens.

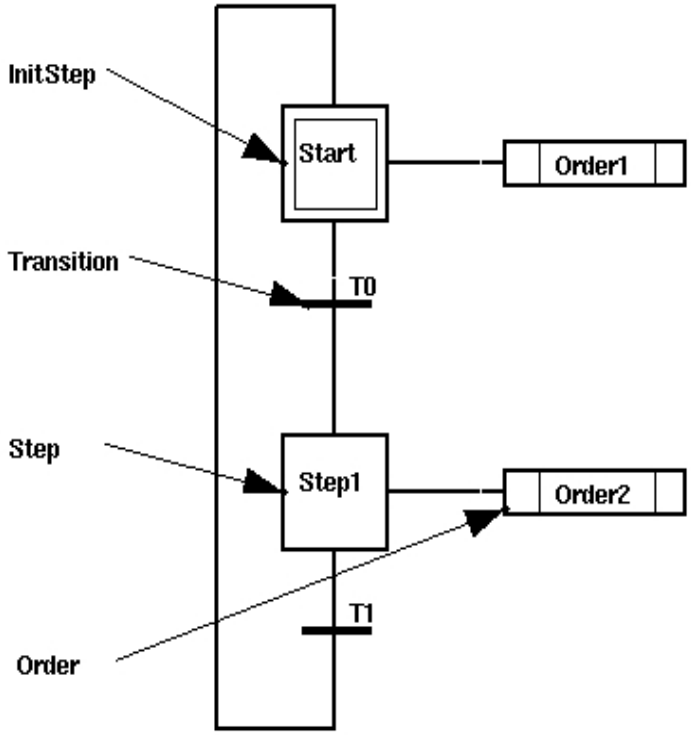


Fig En enkel rak Grafcet sekvens

Förgrenad sekvens

En rak sekvens är den enklaste varianten av sekvenser. Ibland behöver man alternativa grenar i programmet, till exempel om man har en maskin som kan tillverka tre olika produkter. På de ställen som produktionen skiljer sig åt, lägger man in alternativa grenar.

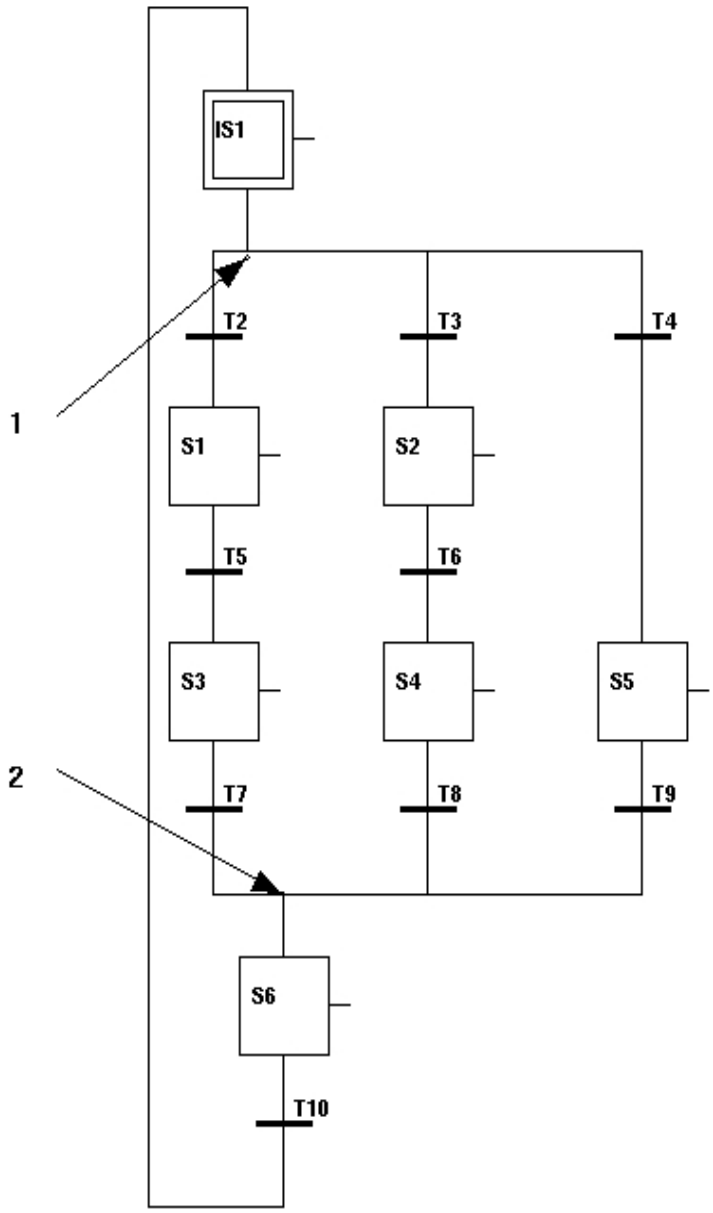


Fig Aternativ förgrening

Exemplet ovan visar sekvensen för en maskin som kan producera tre produkter, röd, grön och blå. Vid förgreningen, punkt 1 i figuren, väljer man önskad gren beroende på den produkt som ska tillverkas. Alternativa grenar utgår från ett steg, som följs av ett övergångsvillkor i varje gren. Det är konstruktörens uppgift att se till att endast ett av övergångsvillkoren är uppfyllt. Om flera skulle vara uppfyllda avgör slumpen vilken gren som väljs. Vid punkt 2 i figuren går grenarna ihop till ett gemensamt steg.

Parallella sekvenser

Ibland behöver man starta flera parallella arbetsmoment samtidigt. Det ska vara möjlig för de olika momenten att kunna arbeta oberoende av varandra. För att åstadkomma detta, använder man parallella sekvenser.

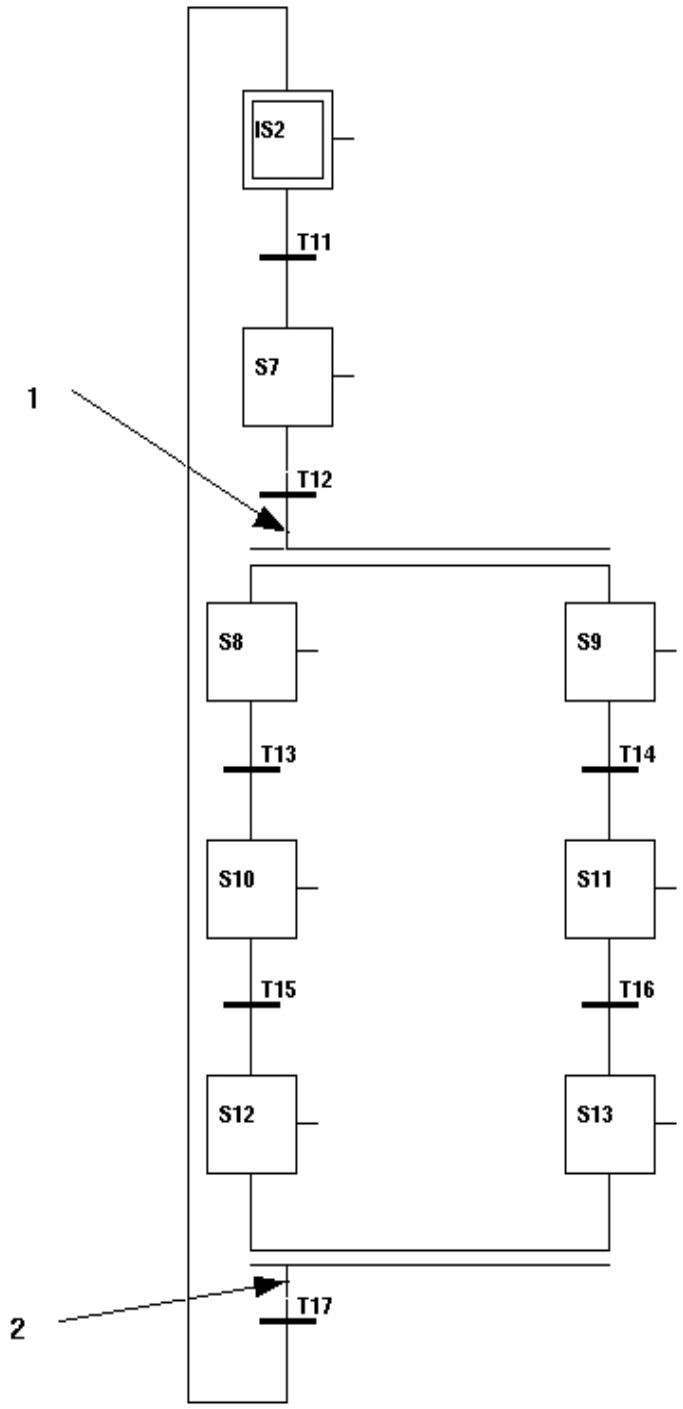
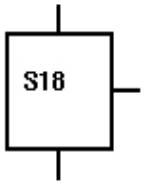


Fig Parallell sekvens

Exemplet i figuren ovan visar sekvensen för två maskiner som borrar två hål samtidigt, oberoende av varandra. När övergångsvillkoret före parallellförgreningen (punkt 1 i figuren), är uppfyllt, flyttas aktiviteten över till båda grenarna, och de två maskinerna börjar borra. Borrningen sker oberoende av varandra.

Grenarna sammanförs till ett övergångsvillkor (punkt 2 i figuren), och när borrningen är klar i båda maskinerna, dvs både S12 och S13 är aktiva, och övergångsvillkoret T17 är uppfyllt, flyttas aktiviteten vidare till initsteget IS2.

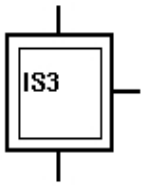
Steg - Step



Ett Step objekt (steg) används för att beskriva ett tillstånd i processen. Följande gäller för ett steg:

- Ett steg kan vara aktivt eller inaktivt. Attributet Order[0] innehåller aktiviteten.
- Man kan koppla en eller flera order till ett steg.
- Steget kan ges ett godtyckligt namn.
- Steget blir inaktivt om resetsignalen sätts.

Initsteg - InitStep



I varje sekvens måste det finnas ett initsteg (InitStep) som skiljer sig från vanliga steg på följande punkter

- Man bör endast ha ett initsteg i en sekvens.
- När programmet startar, är initsteget aktivt.
- Initsteget blir aktivt om resetsignalen sätts.

Övergångsvillkor - Trans

Ett övergångsvillkor används för att överföra aktiviteten mellan två steg. Aktiviteten överförs från det övre till det undre steget (se figur nedan). Ett logiskt villkor, t ex en digital signal, som kopplas till Trans objektet, avgör när överföringen sker.

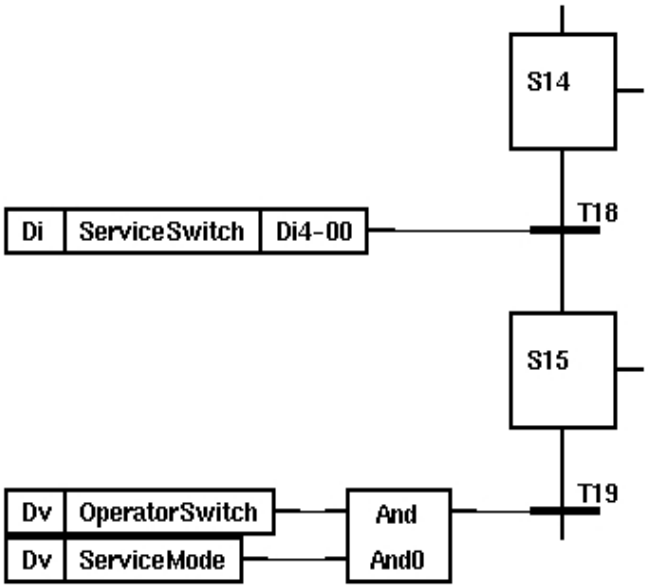


Fig Exempel på övergångsvillkor

Order

Det är möjlig att koppla en eller flera order till varje steg.

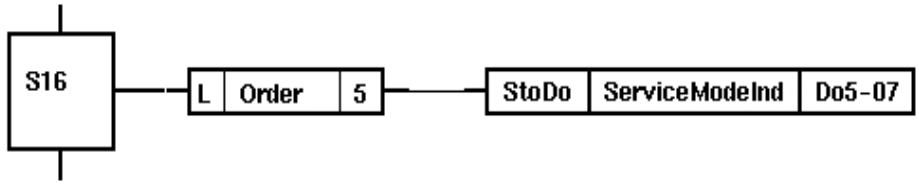


Fig Exempel på order

Normalt är orderns utgång aktiv när ingången är aktiv, men för varje order finns ett antal attribut, som påverkar utgångens funktion:

- D Fördröjning (delay)
- L Tidsbegränsning (limit)
- P Puls (pulse)
- C Villkorlig (conditional)
- S Lagrad (stored)

Dessa funktioner beskrivs i detalj i ProviewR Objekt handbok. Principen är att man anger namnet på attributet (stor bokstav) och eventuell tid i objektseditorn. I figuren nedan visas hur man fördröjer orderns aktivitet i 10 sekunder.

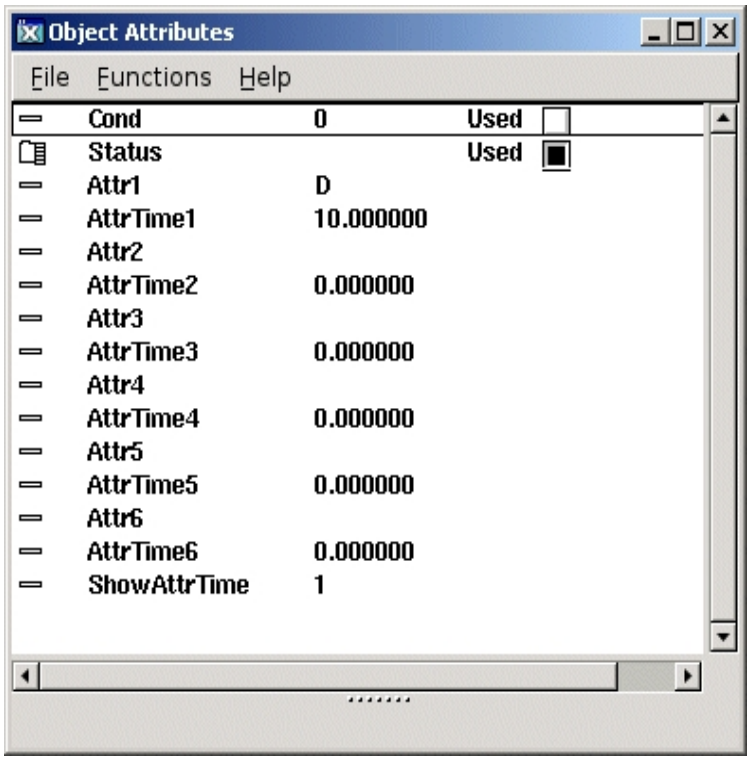


Fig Attribut för DOrder

De valda attributen skriv ut i ordersymbolen.

Figuren nedan visar hur man kan använda en order med fördröjning för att göra ett steg aktivt en viss tid.

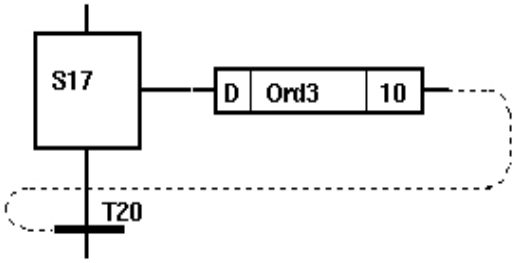


Fig Fördröjd övergång

Notera att man måste använda en feedback förbindning för att ansluta den fördröjda ordern till övergångsvillkoret. Annars blir exekveringsordningen tvetydig.
Se feedback koppling

Subsekvens - SubStep

När man skapar komplexa Grafcet program, är det ofta en fördel att använda underfönster, och i dessa placera subsekvenser. På det här sättet får man en bättre layout på programmet.

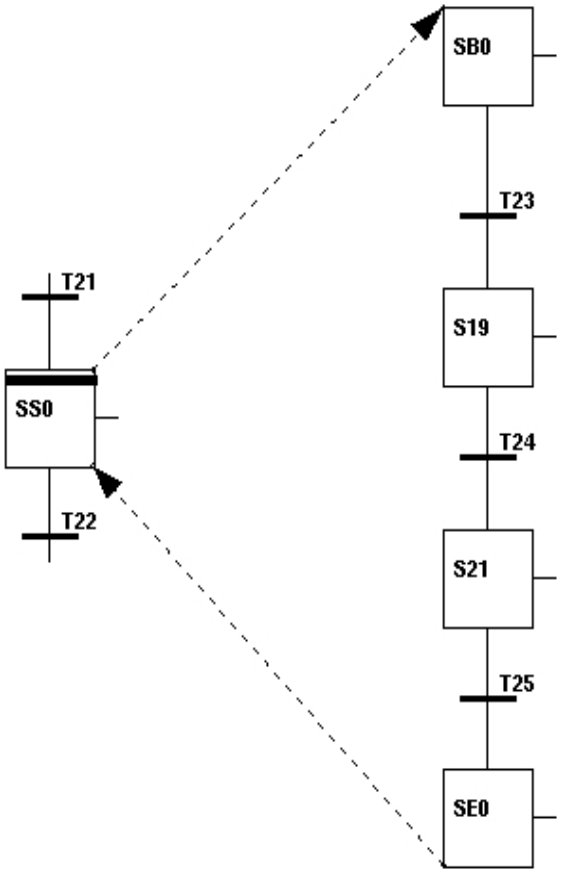


Fig Subsekvens

I figuren ovan visas subsekvensen för ett SubStep. En subsekvens startar alltid med ett SsBegin objekt och slutar med ett SsEnd objekt. En subsekvens kan i sin tur innehålla subsekvenser.

Att bygga Grafcet sekvenser

Grafcet sekvenser byggs enkelt genom att starta med ett Step eller InitStep objekt.
Genom att dra ut en koppling från den nedre kopplingspunkten, och släppa den i ett tomt område i arbetsarean, skapas ett förbundet Trans objekt. På samma sätt skapas Order objekt från den högra kopplingspunkten, och från Trans objektets övre och undre kopplingspunkter skapas nya Step objekt.

Introduktion till funktionsblocks programmering

Block för att hämta och lagra värden

Hämtnings- och lagringsblock används för att läsa och skriva värden. Det finns hämtnings- och lagringsblock för varje typ av signal. I figuren nedan visas ett antal av dessa block.
De återfinns under 'Signal' mappen i paletten.

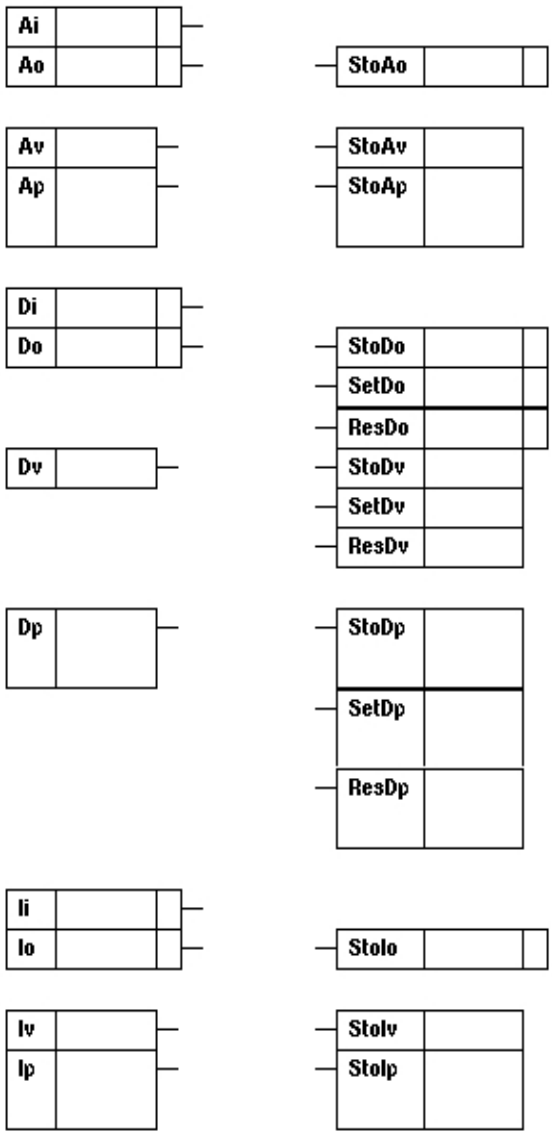


Fig Block för att hämta och lagra värden

För att läsa signalvärden används block som GetAi, GetIi, GetDi eller GetAo. När man vill sätta ett värde i en signal, används block som StoAv, StoDo, SetDv eller ResDo.

Digitala värden kan skrivas på två sätt:

- 'Sto' lagrar ingångsvärdet, dvs om ingången är 1 blir signalen 1, och om ingången är 0 blir signalen 0.
- 'Set' sätter signalen till 1 om ingången är sann, 'Res' sätter ingången till 0 om ingången är sann. Om man, t ex, sätter en digital utgång med ett SetDo objekt, förblir den satt tills man återställer den med ett ResDo objekt.

För att läsa resp. tilldela värden på attribut (andra än ActualValue) använder man följande:

- GetAp och StoAp för analoga attribut.
- GetIp och StoIp för heltals attribut.
- GetDp och StoDp, SetDp eller ResDp för digitala attribut.
- GetSp för strängattribut.
- GetAtp och GetDtp for tidsattribut.

Logiska block

Det finns ett antal block tillgängliga för logisk programmering. t ex och-grind (And), eller-grind (Or), inverterare och timer. För logisk programmering används digitala signaler. Objekten är placerade under mappen 'Logic' i paletten.

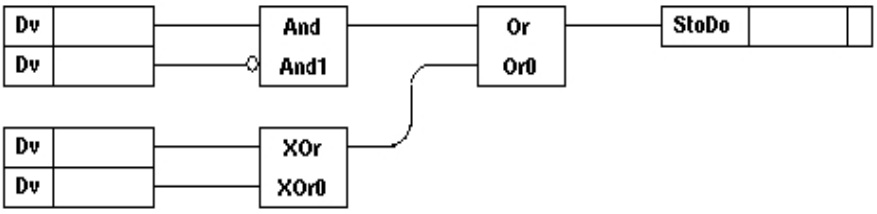
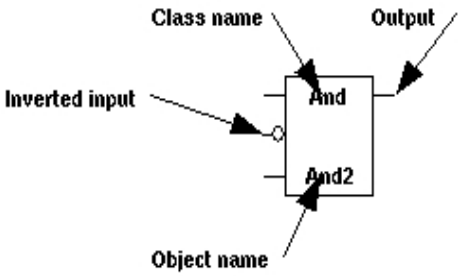


Fig Logiska block

I figuren ovan ses en och-grind (And1). För objektet gäller följande:

- Ingångar till vänster
- Utgångar till höger
- Klassnamnet utskrivet överst.
- Objektsnamnet utskrivet underst (kan ändras av konstuktören).
- Man kan använda ett variabelt antal ingångar, default är 2 st.
- Ingångarna kan inverteras, vilket markeras med en cirkel på ingången.

Och-grind



Och-grindens attribut ändras från objektseditorn.

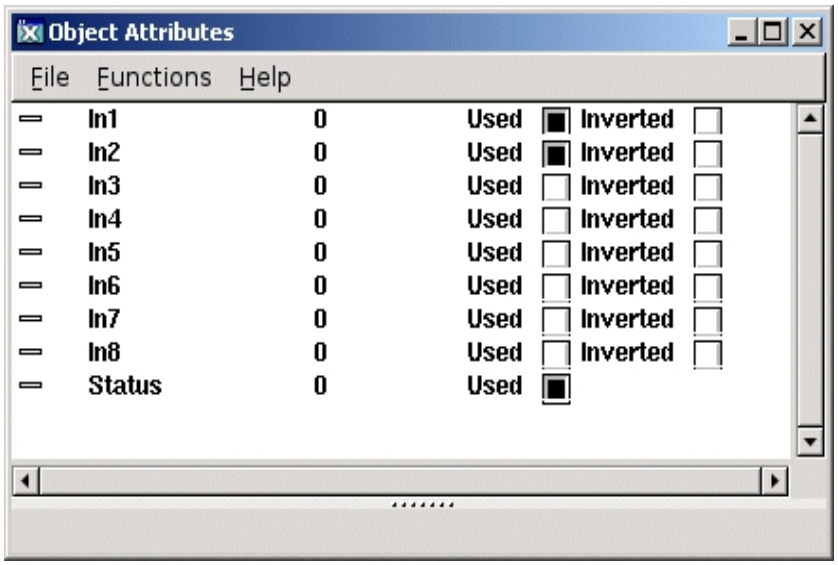


Fig Och-grindens attribut

De andra objekten under 'Logic' mappen har liknande attribut, se ProviewR Objekt handbok.

Beräkningsblock

Mappen 'Analog' i paletten innehåller ett antal objekt för att hantera analoga signaler, t ex filter, additionsblock och integratorer.

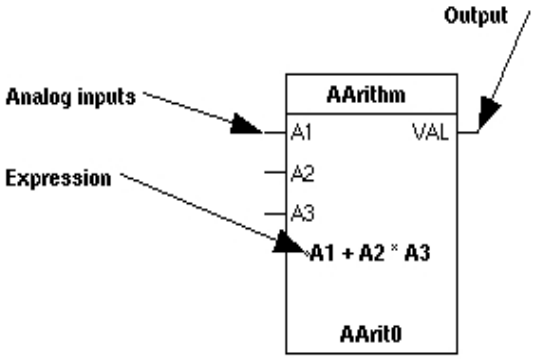


Fig Aritmetiskt beräkningsblock

Här beskrivs inte funktionen hos objekten, men det kan vara lämpligt att kommentera användningen av aritmetiska block. Blocken används för beräkning av användardefinierade uttryck, som skrivs i programmeringsspråket C.

I figuren ovan kommer blocket att beräkna värdet (A1 + A2 * A3) och lägga detta på utgången. A1, A2 och A3 representerar analoga värden, t ex signaler som antas vara kopplade till ingångarna på objektet.

Uttrycken kan innehålla avancerad C kod med arrayer och pekare. När man skriver dessa bör man medveten om att indexering utanför arrayer, eller felaktiga pekare kan gör att exekveringen av plcprogrammet stoppas.

Larmövervakning

I ProviewR är det möjligt att övervaka analoga och digitala signaler. Övervakningen av analoga signaler sker mot ett gränsvärde. Om gränsvärdet överskrids, skickas ett larm till 'event monitorn' som i sin tur skickar det vidare till en utenhet, t ex ett larmfönster.

Se ProviewR Objekt handbok beträffande objektens attribut.

Övervakning av digitala signaler

För att övervaka digitala signaler eller attribut, använder man DSup objekt (Digital Supervisory), som finns under mappen 'Logic' i paletten.

Signal eller ett attributet som ska övervakas, hämtas med ett Get-objekt som förbinds med DSup objektet. Utgångar på logiska block kan kopplas direkt till DSup objektet.

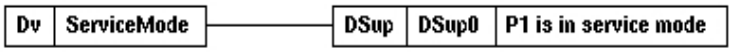


Fig Digital övervakning

Figuren ovan visar övervakningen av en Dv signal.

Attributet 'CtrlPosition' i DSup objektet indikerar om larmet ska aktiveras när den övervakade signalen blir sann eller falsk.

Övervakning av analoga signaler

För att övervaka analog signaler eller attribut, använder man ASup objekt (Analog Supervisory), som finns under mappen 'Analog' i paletten.

Övervakningen sker på liknande sätt som för en DSup, med undantaget att man kan välja om larmet ska utlösas om gränsvärdet underskrids eller överskrids.

Exekveringsordning

Exekveringsordningen bestäms normalt efter hur funktionsobjekten är sammankopplade. Ett objekt vars utgång är kopplad till en ingång på ett annat objekt, exekverar före det objekt det är kopplat till. Man kan se exekveringordningen genom att aktivera 'View/Show Execute Order' i plc-editorns meny. För objekt och nät som inte är sammankopplade exekveras bestämmer koordinaterna exekveringsordningen. Objekt som ligger högre upp exekveras först.

Om man vill påverka exekveringsordningen mellan två objekt kan man koppla en speciell exekveringsordnings koppling mellan objekten. Denna väljs i kopplings-paletten och har en pil i ena ändan. Den fungerar så att objektet som kopplingen pekar på exekverar efter objektet som kopplingen utgår från.

Om ett nät innehåller en återkoppling, kan exekeveringordningen inte bestämmas och man får felet 'Ambigous execute order'. Man måste då byta ut en koppling mot en feedback koppling, dvs en koppling som inte bestämmer exekveringsordningen, så att exekveringsordningen blir entydigt bestämd. Feedback-kopplingen är sträckad och väljs i verktygs-panelen eller i kopplings-paletten.

I/O kopiering

Om en signal används på flera ställen i ett nät av funktions-objekt, och det finns risk att signalens värde ändras under exekveringen kan man få låsningar och andra fenomen som är svåra att förutse. Därför används en mekanism som kallas I/O kopiering. Signalvärdena för signaler av typen Ai, Ao, Av, Di, Do, Dv, Ii, Io, Iv, Co, Bo och Bi, är samlade i speciella area-objekt. Innan en plc-tråd börjar exekvera tar man en kopia av areaobjekten, och under exekveringen görs alla läsningar från kopian, medan skrivningar görs i area-objekten.
Det innebär dels att man undviker den här typen av fenomen, men också att man kan få fördröjningar. Om man sätter värdet på en signal, och sedan läser det i samma plc-tråd, kommer förändringen inte att registreras förrän i nästa skan.

Kompilera plcprogrammet

Innan man börjar kompilera, måste man ange vilken platform (eller vilka plattformar) som plcprogrammets volym ska exekvera på. Öppna volymsattribut-editorn från konfiguratören, 'File/Volume Attributes' i menyn, och ange operativsystem. Notera att mer än ett operativsystem kan väljas. Volymen kan samtidigt köras i produktions-systemet, på ett simulerings-system och i ett utbildnings-system, och alla kan ha olika plattform.

Nu kompileras plcprogrammet genom att välja 'File/Build' i plceditorns meny. Eventuella fel och varningar visas i meddelandefönstret. Även när man bygger noden, kommer alla nya eller modifierade plcpgm att kompileras.

11 Anropa funktioner från plc programmet

Den funktions-blocks programmering som plc-programmet utgörs av har sina begränsningar, och vissa saker kan göras mycket enklare och snyggare med c-kod. För detta finns funktionsblocken CArithm och DataArithm där man kan lägga in en viss mängd c-kod, men antal tecken är 1023 (8191 för DataArithmL), och ibland räcker det inte till. Man har då två möjligheter, att skriva en fristående applikation, vilket finns beskrivet i kapitlet Applikationsprogrammering, eller att anropa en c-funktion från en CArithm eller DataArithm. Fördelen med att anropa en funktion är att all initiering och knytning till objekt sköts av plc-programmet. Dessutom sker anropen synkront med exekveringen av den plc-tråd som anropar funktionen.

Skriva koden

Koden läggs i en c-fil, som läggs någonstans under $pwrp_src. Vi skapar filen $pwrp_src/ra_myfunction.c och lägger in funktionen MyFunction() som utför en enkel beräkning.

```
#include "pwr.h"
#include "ra_plc_user.h"

void MyFunction( pwr_tBoolean cond, pwr_tFloat32 in1, pwr_tFloat32 in2,
                 pwr_tFloat32 *out)
{
  if ( cond)
    *out = in1 * in2;
  else
    *out = in1 + in2;
}
```

Prototypdeklaration

I includefilen ra_plc_user.h läggs en prototypdeklaration av funktionen.

```
void MyFunction( pwr_tBoolean cond, pwr_tFloat32 in1, pwr_tFloat32 in2,
                 pwr_tFloat32 *out);
```

ra_plc_user.h includeras av plc-programmet, och funktionen kan därmed anropas från CArithm eller DataArithm objekt. Man bör även includera ra_plc_user.h i funktions-koden för att få en kontroll på att anropet stämmer.

ra_plc_user bör ligga på $pwrp_src och kopieras till $pwrp_inc varifrån den inkluderas av plc't och funktions-kod.

Kompilera koden

c-filen kompileras, till exempel med hjälp av make. Nedan visas en makefile som kompilerar ra_myfunction.cpp och lägger resultatet, objektsmodulen ra_myfunction.o på $pwrp_obj. Notera att även ett beroende på ra_plc_user.h har lagts in så att denna kommer att kopiera

från $pwrp_src till $pwrp_inc.

```
ra_myfunction_top : ra_myfunction

include $(pwr_exe)/pwrp_rules.mk

ra_myfunction_modules : \
  $(pwrp_inc)/ra_plc_user.h \
  $(pwrp_obj)/ra_myfunction.o

ra_myfunction : ra_myfunction_modules
 @ echo "ra_myfunction built"

#
# Modules
#

$(pwrp_inc)/ra_plc_user.h : $(pwrp_src)/ra_plc_user.h

$(pwrp_obj)/ra_myfunction.o : $(pwrp_src)/ra_myfunction.c \
                              $(pwrp_inc)/ra_plc_user.h
```

Anrop från plc-programmet

Anropet av funktionen sker från ett CArithm, DataArithm, AArithm eller DArithm objekt.

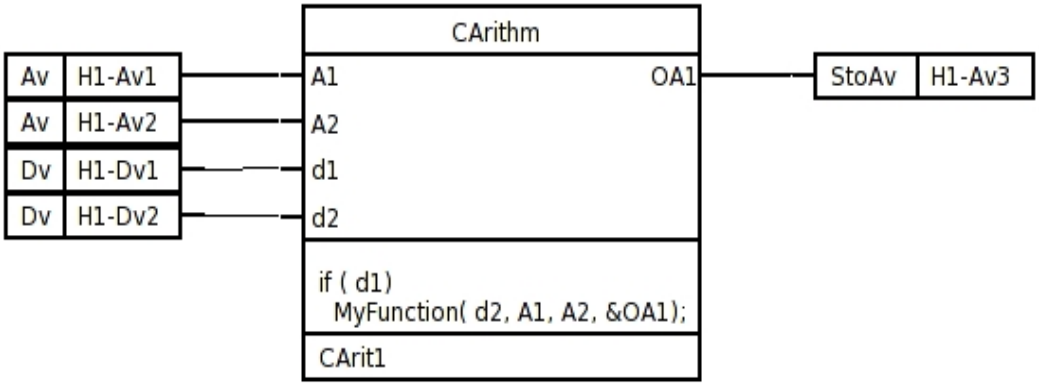


Fig Anrop av funktion från plc-koden

Länka plc-programmet
Vid kompileringen skapades objektsmodulen $pwrp_obj/ra_myfunction.o. Denna måste länkas med när plcprogrammet byggs, och det sker genom att skapa ett BuildOptions objekt i directory volymen under NodeConfig objektet. Lägg in namnet på objektsmodulen i ObjectModule vektorn. När direktory volymen sparas genereras en opt-fil på $pwrp_exe som inkluderas av länkaren när noden byggs.

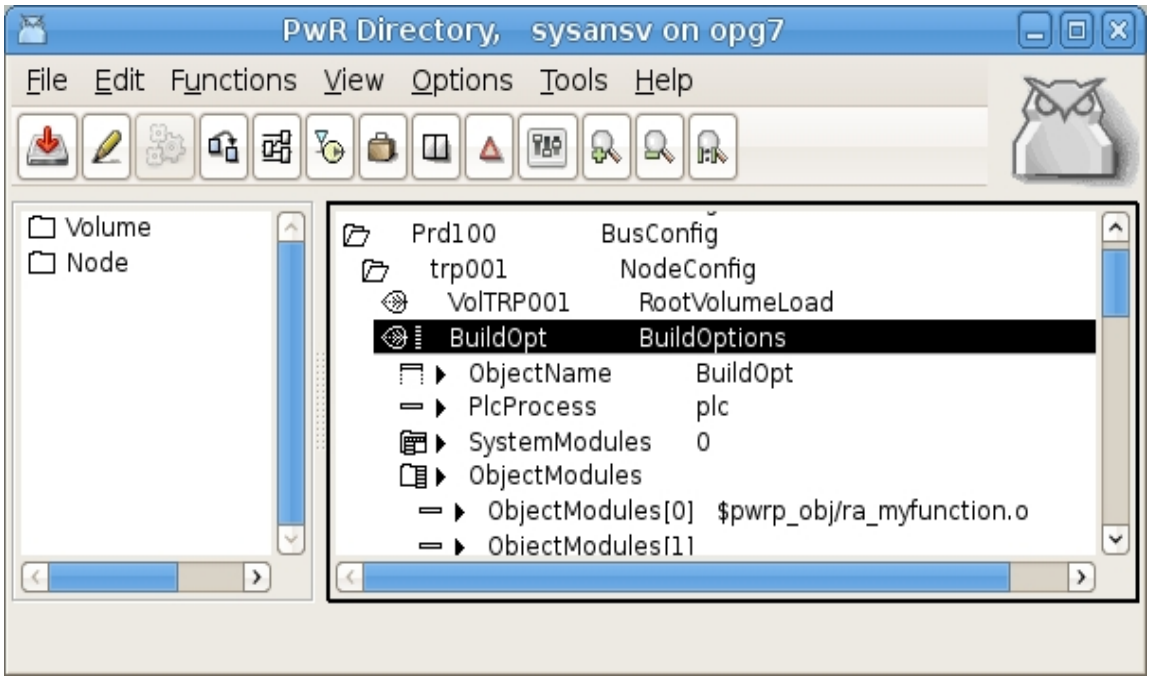


Fig Objektsmodulen inlagd i BuildOptions

Vi kan nu bygga noden och starta ProviewR runtime.

Debug

En nackdel när man lämnar den grafiska programmeringen och anropar c-funktioner är att man inte längre kan följa förloppet i trace. Om man misstänker att det är något fel i koden är ofta bästa sättet att starta plc-programmet i debug, sätta en brytpunkt i funktionen, och stega sig fram i koden.

Först bygger man plc-programmet med debug genom att från konfiguratören öppna Options/Setting i menyn och aktivera Build/Debug, och sedan bygga noden.

Därefter startar man ProviewR runtime och knyter upp debuggern, gdb, mot plcprocessen genom att skicka med pid för processen till gdb med -p. pid kan man se med 'ps x'

```
> ps x
...
5473 pts/0    Sl     0:18 plc_mynode_0999_plc
```

5473 är pid för plcprogrammet. Vi startar debuggern, sätter en brytpunkt i funktionen och låter programmet exekvera vidare

```
> gdb -p 5473 plc_mynode_0999_plc
(gdb) b MyFunction
(gdb) c
```

När programmet går in i funktionen fastnar det i debuggern och vi kan stega (s) och titta på innehållet i variabler (x) mm.

Om plcprogrammet terminerar direkt efter start, kan man starta om det i debug.

```
> gdb plc_mynode_0999_plc
```

Det går också bra att döda den gående plc-processen och starta en ny i debug

```
> killall plc_mynode_0999_plc
> gdb plc_mynode_0999_plc
```

12 Komponenter och Aggregat

Det här kapitlet handlar om hur man programmerar med hjälp av komponenter och aggregat.

En komponent är ett (eller att antal objekt) som hanterar en komponent i anläggningen. En komponent kan var t ex en ventil, en kontaktor, en temperaturgivare eller en frekvensomformare. Eftersom det här är komponenter som förekommer i många olika typer av anläggningar är det idé att försöka göra ett objekt som innehåller allt som behövs för att styra och kontrollera komponenten, och som är så generellt att det kan användas i de flesta anläggningar.

En komponent i ProviewR kan vara uppdelat på ett antal objekt:
- ett huvudobjekt som innehåller konfigureringsdata och data som behövs vid övervakning och drift av komponenten. Det innehåller även signal-objekten för komponenten.
- ett funktionsobjekt som läggs in i plcprogrammet och som innehåller koden för att styra komponenten.
- ett I/O objekt som definierar eventuell kommunikation med t ex en profibus modul.
- ett simuleringsobjekt för att enkelt kunna testa och simulera systemet.

Dessutom ingår en objektbild, dokumentation, ev trendkurvor mm.

Aggregat är en större del i anläggningen än en komponent, och innehåller ett antal komponenter. Ett aggregat kan t ex vara en pumpstyrning, som består av komponenterna pump, motor, kontaktor och säkerhetsbrytare. I övrigt är aggregatet uppbyggt som en komponent med huvudobjekt, funktionsobjekt, simuleringsobjekt, objektsbild, dokumentation mm.

Objektsorientering
ProviewR är ett objektsorienterat system, och komponenter och aggregat är ett område där man utnyttjar objektsorienteringens egenskaper fullt ut. I komponenter kan man se hur man bygger upp objekt av andra objekt, att ett attribut förutom att vara en enkel typ som en float eller en integer, även kan vara ett objekt som i sin tur består av andra objekt. Ett attribut som är ett objekt kallas för ett attributobjekt. Det är inte riktigt analog med ett fristående objekt eftersom det inte har något objektshuvud och inte någon egen objektsidentitet, men i övrigt innehåller det alla egenskaper som ett fristående objekt har i form av metoder, objektbilder mm.

Ett exempel på attributobjekt kan vi se i komponentobjektet för en magnetventil. Här ligger signalobjekten, två Di objekt för gränslägen och ett Do objekt för styrsignal för att öppna ventilen, intern i objektet som attributobjekt. Vi behöver alltså inte skapa signalerna separat, utan i och med att ventilobjektet är skapat, har vi även skapat signalerna för detta objekt. Ett annat exempel på attributobjekt är ett motoraggregat som innehåller komponentobjekt för frekvensomformare, säkerhetbrytare, motor mm i form av attributobjekt.

En annan viktig egenskap inom objektsorienteringen är arv. Med arv menas att man kan skapa en subklass utgående från en existerande klass, en superklass. Subklassen som ärver alla egenskaper från superklassen, men man har möjlighet att lägga till eller förändra en del egenskaper. Ett exempel på detta är en komponent för en temperaturgivare som är en subklass till ett generellt givarobjekt för analoga givare. Den ända skilladen mellan temperatur-givarklassen och dess superklass är objektsbilden, där givarvärdet presenteras i form av en termometer i stället för en stapel. Ett annat exempel är ett pumpaggregat som subklassar ett motoraggregat. Pumpaggregatet har utökats med ett pump objekt och har även modifierade objektsbilder som förutom motorstyrningen även visar en pump.

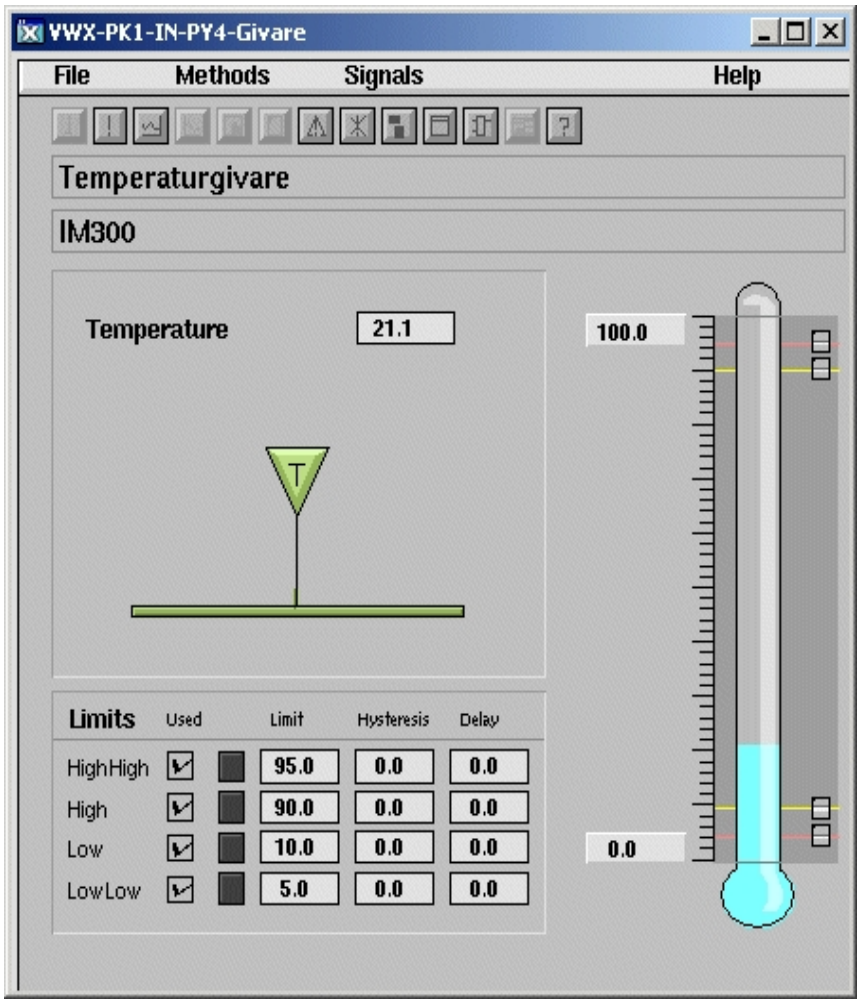


Fig Objektsbild för subklassen BaseTempSensor med superklassen BaseSensor

En annan egenskap som vi har infört är möjligheten att sätta attribut ur funktion. Orsaken till detta är att komponentobjekten måste göras så generella att de täcker in alla varianter av den komponent i anläggningen de ska styra. En magnetventil kan t ex ha ett gränsläge som markerar att ventilen är öppen, men det finns även ventiler med gränslägen som markerar stängd ventil, eller ventiler med båda gränslägena eller ventiler helt utan gränslägen. Man skulle naturligvis ha kunna göra fyra olika komponentklasser, en för varje gränslägesvariant, men problemen uppstår när man vill bygga aggregat av komponenterna. Antalet varianter av ett aggregat blir snart ohanterbart om man ska täcka in alla olika varianter av delkomponenterna. Om vi t ex ska göra ett aggregat som innehåller fyra magnetventiler och varje ventil kan finnas i fyra varianter, blir det 64 varianter av aggregatet. Vill vi dessutom bygga ett aggregat som innehåller fyra ventilaggregat är vi uppe i 4096 olika klasser. Lösningen är att bygga en ventilkomponent med båda gränslägena, där man sedan genom en konfigurering kan sätta attributen för ett eller för båda gränslägena ur funktion, så den kan hantera alla fyra gränslägesvarianterna. I det här fallet är det attributobjekt av klassen Di som sätts ur funktion, vilket innebär att de inte visas i navigatören och ignoreras av I/O hanteringen. Dessutom tas det hänsyn till detta i koden för ventilkomponenten och i objektsbilder. Konfigureringen gör i konfiguratören från popupmenyn där man under 'ConfigureComponent' kan välja en konfigurerring av de olika alternativ som finns.

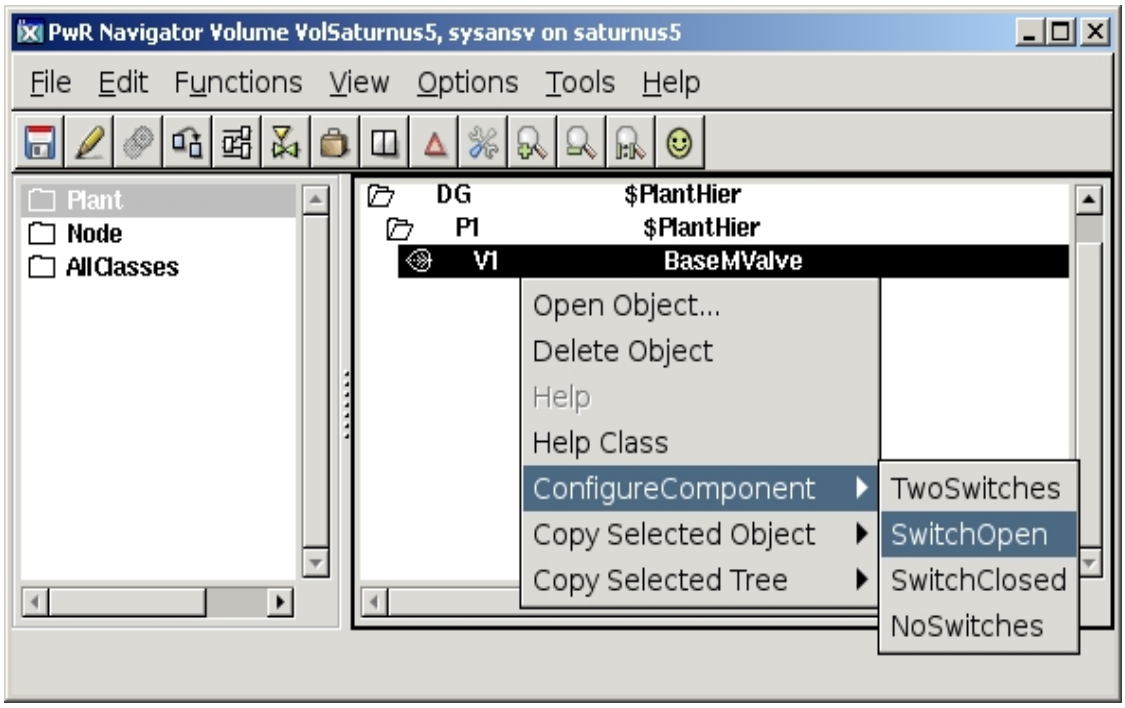


Fig ConfigureComponent metoden för en BaseMValve.

Baskomponenter

ProviewR innehåller ett antal komponent och aggregat objekt för vanliga element i en anläggning, t ex temperaturgivare, tryckgivare, tryckvakt, magnetventil, filter, motorstyrningar och fläktstyrningar. De finns samlade i klassvolymen BaseComponent. En baskomponent kan användas rakt av, och det är nog det vanligaste sättet att använda dem, men tanken är också att man utgående från basklasserna skapar klasser och bygger upp bibliotek med för de specifika komponenter man använder i sina anläggningar.

För magnetventiler finns basklassen BaseMValve. Har man en magnetventil av typen Durholt 100.103 skapar man en subklass med BaseMValve som superklass, Durholt_Valve_100_103, och lägger in den konfigurering som gäller för just den här ventilen, dessutom lägger man in en länk till datablad och fyller i Specification attributet så att man kan identifiera och beställa reservdelar till ventilen. När man använder ett Durholt_Valve_100_103 objekt behöver man sedan inte göra så mycket konfigureringar och anpassningar eftersom detta är gjort redan i klassen. På det här sättet kan man bygga upp komponentbibliotek för de olika typer av komponenter som man använder i sina anläggningar.

Ett problem uppstår när man använder aggregat. Aggregaten innehåller baskomponenter från BaseComponent och finns det specifica subklasser för komponenterna, vill man kunna använda dessa även i aggregaten. Lösningen på problemet är Cast (forma) funktionen. En baskomponent i ett aggregat kan castas till en subklass, under förutsättning att subklassen har samma storlek, dvs att subklassen inte har utökats med nya attribut. Castningen innebär att komponenten hämtar initialvärden, konfigureringar, metoder, objektbilder mm från subklassen, dvs i alla lägen uppträder som den subklass den är castade till. Castningen utförs från popupmenyn i konfiguratören, där man i 'Cast' alternativet får upp en lista på de subklasser som finns att välja på. Genom att välja en subclass castar man komponenten till denna.

Tryckvakt

Låt oss titta på en relativt enkel komponent, en tryckvakt, för att se hur den är uppbyggd och

hur man konfigurerar den. För tryckvakter finns baskomponenten BasePressureSwitch, som är en subklass till BaseSupSwitch. Eftersom temperatur-, tryck-, och nivåvakter är väldigt lika ur övervakningssynpunkt har de en gemensam superklass. BaseSupSwitch har i sin tur en superklass, Component, som är gemensam för all komponentklasser. Klassberoende för tryckvaktsklassen blir med andra ord

```
Component-BaseSubSwitch-BasePressureSwitch
```

Component klassen
Component innehåller attributen Description, Specification, HelpTopic, DataSheet, CircuitDiagram, Note och Photo som alltså finns i alla komponenter. I Description finns plats för en kort beskrivning, i Specification lägger man in modelltypen för den komponent det är frågan om, de övriga används för att konfigurera motsvarande metoder i operatörsmiljön.

BaseSubSwitch
Från superklassen BaseSupSwitch ärvs attributen Switch, AlarmStatus, AlarmText, Delay, SupDisabled och PlcConnect.

- Switch är Di objektet för tryckgivaren. Detta måste kopplas till ett kanalobjekt i nodehierarkin.
- AlarmStatus visar larmstatus i runtime.
- AlarmText innehåller larmtexten för det larm som skickas vi larmstatus. Larmtexten har defaultvärdet "Pressure switch, ", men kan naturnligvis bytas ut mot någonting annat. Observera att om defaulttexten behålls, kommer denna att översättas om man väljer att annat språk. Lägger man in en annan text, sker ingen översättning.
- Delay anger larmets fördröjning i sekunder, (default 0 sekunder).
- SupDisabled indikerar att larmet är undertryckt.
- PlcConnect är koppling till funktionsobjekt i plc-koden.

Till huvudobjektet BaseSupSwitch, finns det ett funktionsobjekt BaseSupSwitchFo, och även detta ärvs av tryckvakskomponenten.

BasePressureSwitch
BasePressureSwitch har inte några attribut utöver de som har ärvts från superklasserna. Det som är unikt för BasePressureSwitch är den grafiska symbolen, Components/BaseComponent/PressureSwitch, och objeksbilden, där switch-symbolen innehåller ett P som markerar tryck.

Konfigurering
Vi öppnar konfiguratören och lägger in huvudobjektet, BasePressureSwitch, i anläggning-hierarkin. I ett lämpligt PlcPgm lägger vi in funktionsobjektet BaseSupSwitchFo och kopplar detta till huvudobjektet med connect-funktionen. Välj ut huvudobjektet och klicka med Shift/Dubbelklick MB1 på funktionsobjektet. Funktionsobjektet innehåller koden för komponenten, som för en BaseSupSwitch är ett larm skickas vid larmtillstånd.

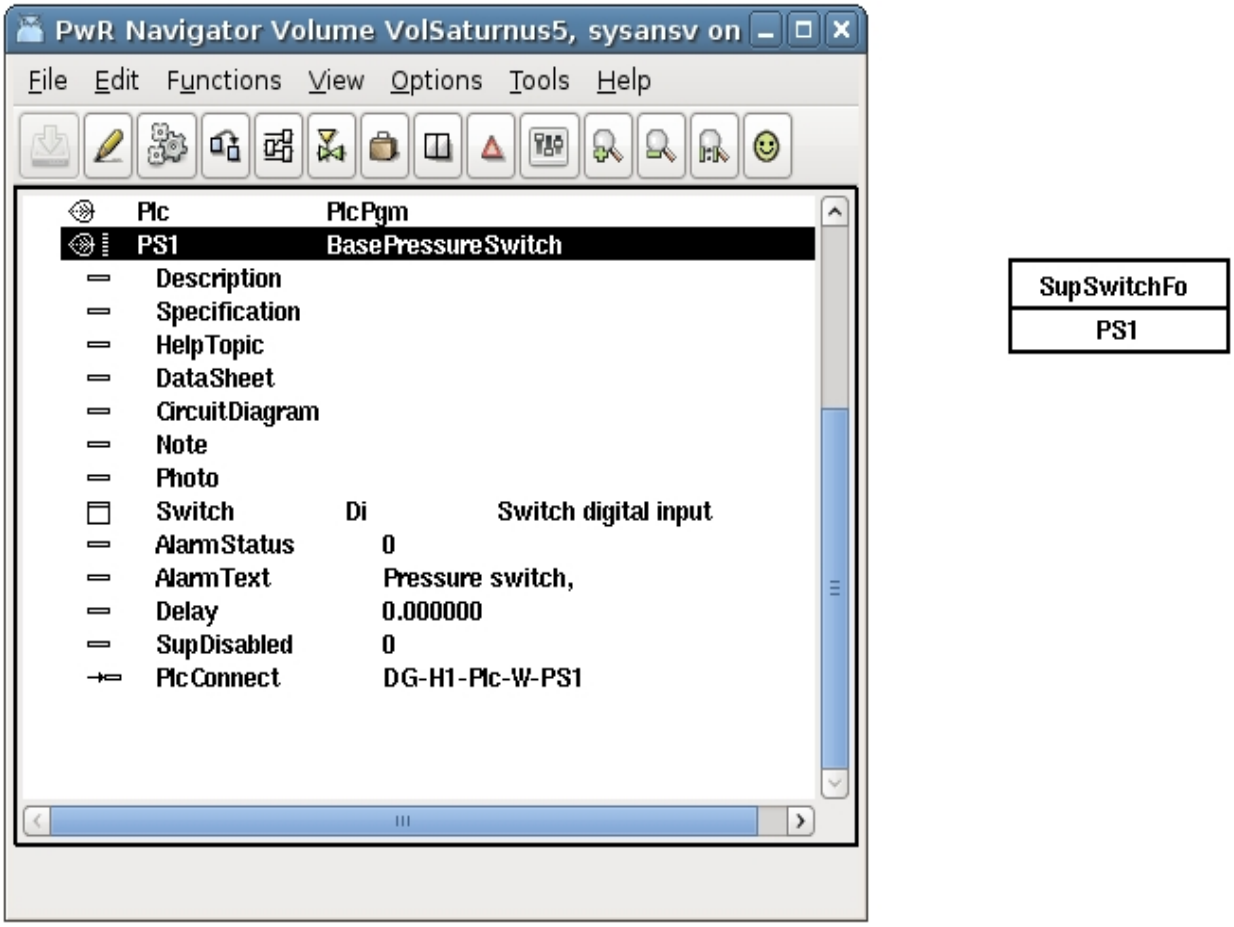


Fig Huvudobjekt med tillhörande funktionsobjekt

Tryckvakten ska visas för operatören i en Ge bild. Vi öppnar Ge-bilden och hämtar subgrafen BaseComponent/SupSwitch från paletten. Subgrafen är anpassad till till en BaseSupSwitch, så det ända vi behöver göra är att koppla den till huvudobjektet. Välj ut huvudobjektet i planthierarkin och klicka med Shift/Dubbelklick MB1 på subgrafen.

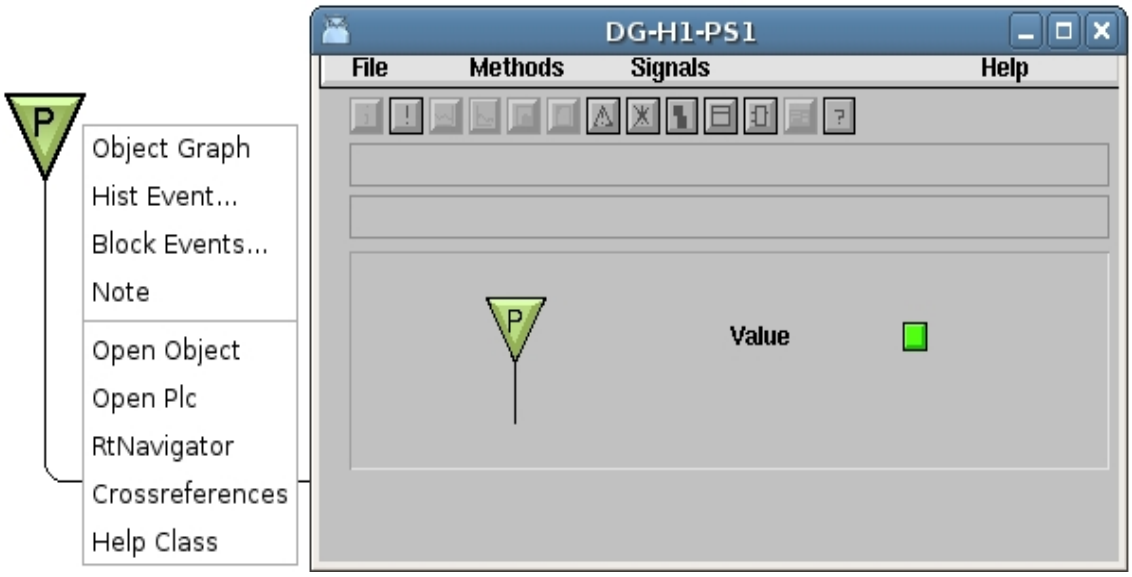


Fig Grafisk symbol, popupmeny med metoder och objeksbild

Det här räcker för att göra komponenten funktionsduglig. Sedan kan man naturligvis forsätta

att konfigura metodattributen med hjälptexter och länkar till fotografier, elschemor och datablad.

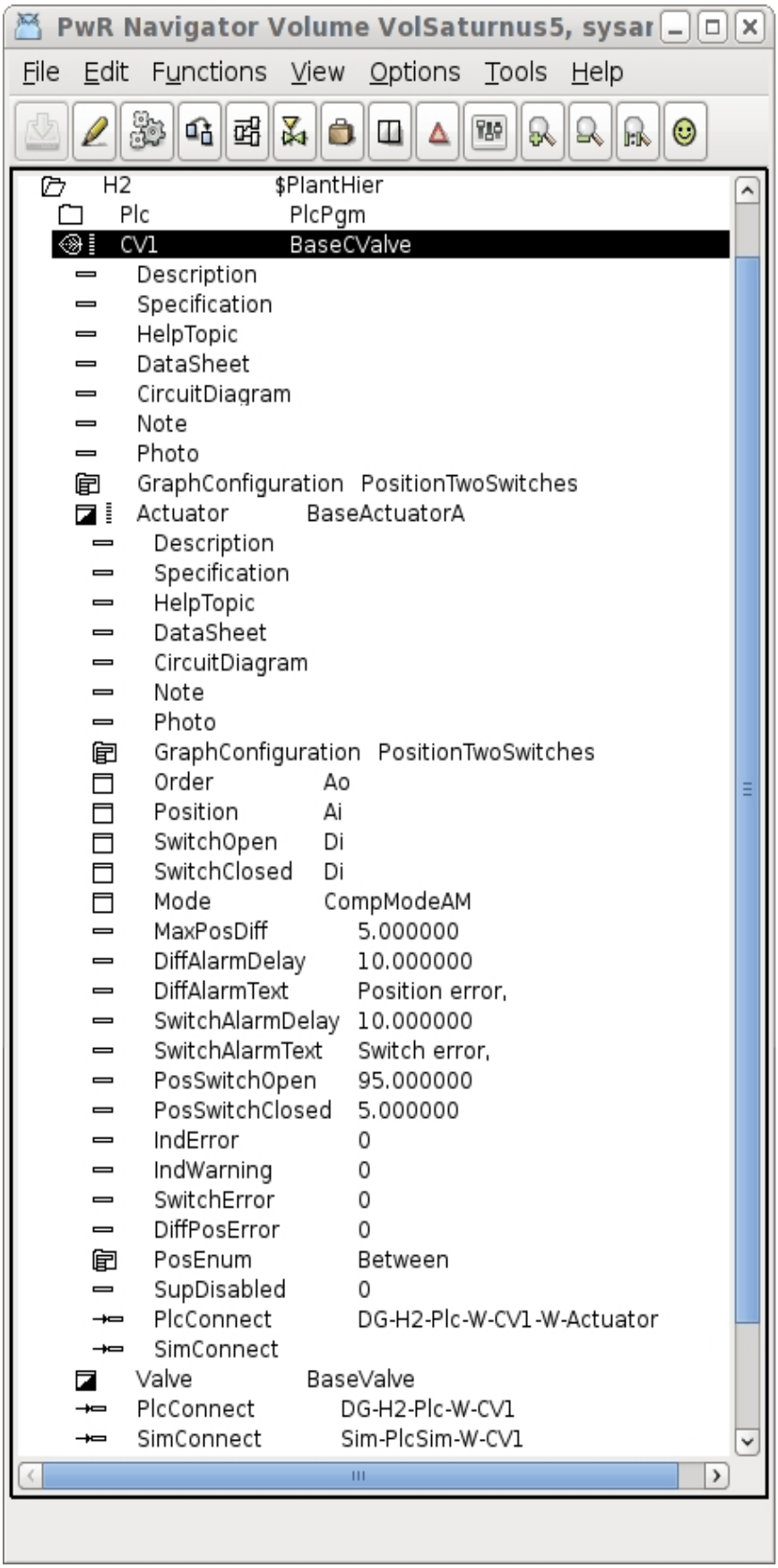
Reglerventil

Nu ska vi titta närmare på en lite mer omfattande komponent, BaseCValve, som styr en reglerventil. Till skillnad från tryckvakten ovan, måste man använda ConfigureComponent för att konfigurerar objektet, dessutom finns det ett simuleringsobjekt som används för att testa systemet.

Antag att vi har en reglerventil, som styrs med en analog utsignal, och som ger positionen tillbaka i en analog ingång. Här kan vi använda en BaseCValve. Den har den analoga utsignalen 'Order' och den analoga insignalen 'Position', dvs de signaler som vi efterfrågar. Dessutom finns två digitala insignaler för gränsläge öppen och gränsläge stängd, men dessa kan elimineras med hjälp av konfigurering.

Konfigurering

Vi lägger in huvudobjektet BaseCValve i planthierarkin och funktionsobjektet BaseCValveFo i ett PlcPgm, och kopplar ihop dem med Connect funktionen. Funktionsobjektet har en orderingång som vi kopplar till ett PID objekt. Vi måste även ange att vår ventil inte har några gränslägen, och det gör vi genom att välja 'ConfigureComponent/PositionNoSwitches' i popupmenyn för huvudobjektet. När vi nu öppnar huvudobjektet, och i detta Actuator objektet, ska vi hitta en Position signal, men inga gränslägessignaler.



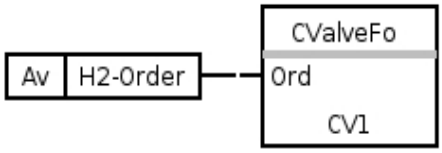


Fig Huvudobjekt med funktionsobjekt

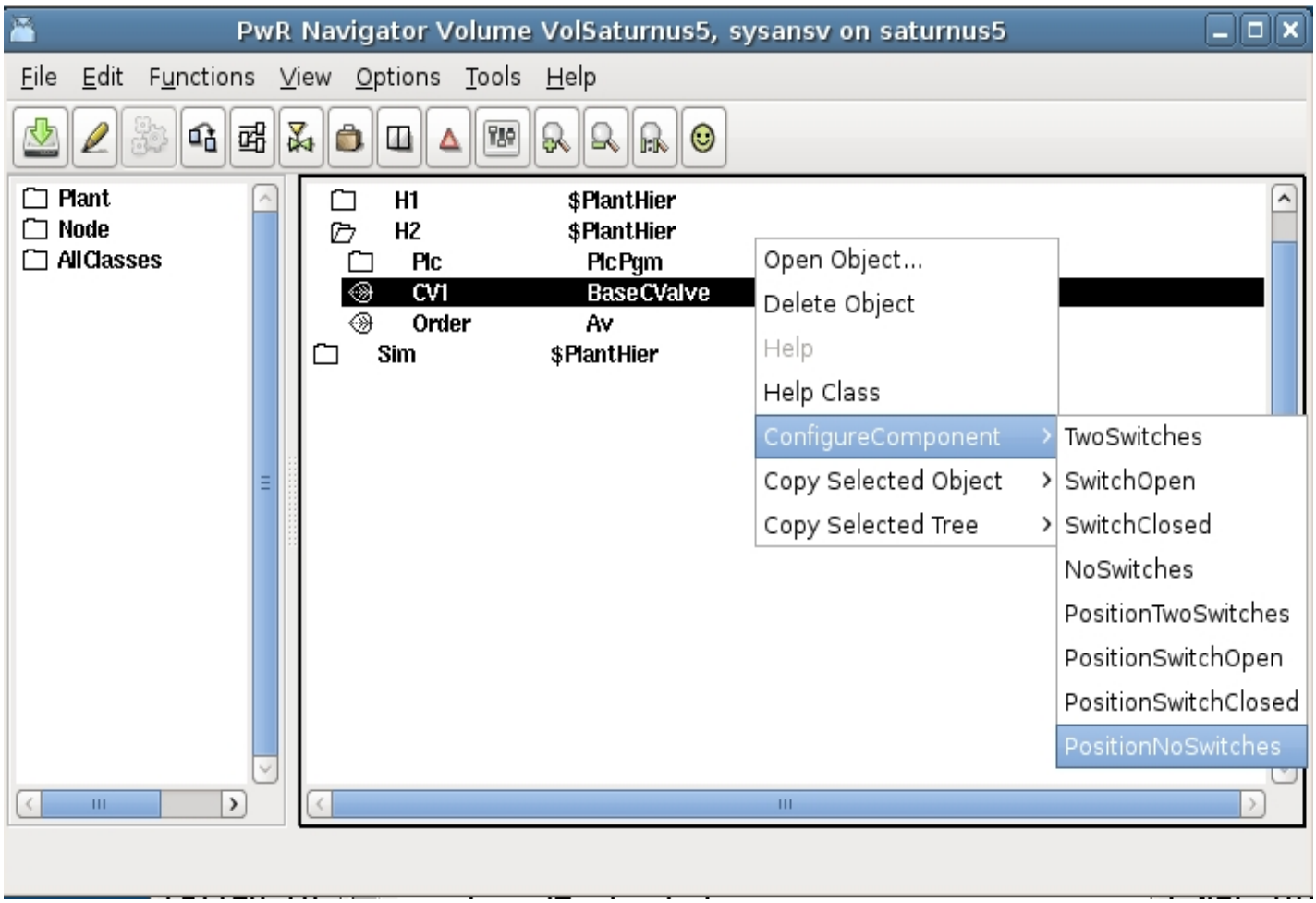


Fig ConfigureComponent alternativet PositionNoSwiches

Vi lägger även in subgrafen BaseComponent/CValve i en Ge-bild och kopplar denna till huvudobjektet.

Nu vill vi även kunna simulera ventilen, för att se att den fungerar som vi har tänkt. För simulering finns funktionsobjektet BaseCValveSim som vi lägger i ett speciellt PlcPgm för simulering, eftersom det inte ska exekvera i driftsystemet. Funktionsobjektet kopplas till huvudobjektet med Connect-funktionen.

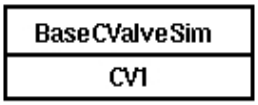


Fig Simuleringsobjekt för reglerventilen

Konfigureringen är klar och efter att ha byggt simuleringsnoden kan vi testköra och titta på resultatet.

Pumpstyrning

Nästa exempel är ett aggregat, en pumpstyrning med en frekvensomformare som kommunicerar via Profibus med protokollet PPO3. Vi kommer att se hur ett komponentobjekt i aggregatet, förutom huvudobjekt, funktionsobjekt och simuleringobjekt, även omfattar I/O objekt för att hämta och skicka data via profibus.

Klassen vi använder är BaseFcPPO3PumpAggr, och om vi tittar på klassberoendet är det

```
Aggregate-BaseFcPPO3MotorAggr-BaseFcPPO3PumpAggr
```

Alla aggregat har superklassen Aggregate som motsvarar Component classen för komponenter. Nästa superklass, BaseFcPPO3MotorAggr innehåller all funktionalititet för styrningen. Själva pump klassen BaseFcPPO3PumpAggr utökar motoraggregatet med ett pumpobjekt som representerar den mekaniska pumpen, men som inte innehåller några signaler eller någon ytterligare funktionalitet. Dessutom har pumpaggregats klassen en egen objektbild och en egen grafisk symbol.

Konfigurering
Huvudobjektet BaseFcPPO3PumpAggr läggs i anläggningshierarkin och funktionsobjektet (som ärvts från motoraggregatet) BaseFcPPO3MotorAggrFo läggs i ett PlcPgm, och de båda kopplas ihop med Connect funktionen.

Huvudobjektet har inte mindre än 24 olika konfigureringar att välja mellan, beroende på vilka av komponenterna Fuse, Contactor, SafetySwich, StartLock och CircuitBreaker som finns med i konstruktionen. I vårt fall har vi bara en kontaktor och en frekvensomformare och väljer alternativet ConfigureComponent/CoFc.

En del komponenter i ett aggregat kan ha egna konfigureringar. I det har fallet kan kontaktorn och motorn konfigureras individuellt. Vår kontaktor har två signaler, en Do för order och en Di för feedback, och det stämmer med default konfigurationen, så denna behöver vi inte ändra på. Motorn däremot har ett motorskydd i form av en temperaturvakt, därför väljer vi ut motor komponenten och aktiverar ConfigureComponent/TempSwitch i popupmenyn.

Nästa steg är att koppla ihop signalobjekt med kanalobjekt. Kontaktorn innehåller en Do och en Di för order resp feedback som kopplas till lämpliga kanalobjekt i nodehierarkin. Motorn, har en temperaturvakt i form av en Di som också ska kopplas. Frekvensomformaren innehåller fyra signaler, StatusWordSW (Ii), ActSpeed (Ai), ControlWordCW (Io) och RefSpeed (Ao). Dessa signaler utbyts med frekvensomformaren via Profibus med protokollet PPO3. Det finns ett speciellt Profibus modulobjekt för PPO3, BaseFcPPO3PbModule, innehåller kanaler för PPO3 kommunikation. Modulen konfigureras mha Profibus konfiguratorn (see Guide to I/O System), och kopplas ihop med FreqencyConverter komponenten i pumpaggregatet. Eftersom komponentobjektet och modulobjektet är anpassade till varandra, behöver man inte koppla signal för signal, utan man kopplar component med modul istället. Genom att välja ut modulobjektet och aktivera ConnectIo i popupmenyn for FrequencyConverter komponenten är kopplingen gjord.

Vi lägger även in subgrafen Components/BaseComponents/FcPPO3PumpAggr i en översiktsbild och kopplar denna till huvudobjektet. Vidare lägger vi in ett simuleringsobjekt BaseFcPPO3MotorAggrSim i ett speciellt simulerings PlcPgm, och kopplar detta till huvudobjektet.

Mode
PumpAggregatet innehåller ett Mode objekt där man konfigurerar varifrån pumpen styrs. Det innehåller moderna Auto, Manuel och Local med följande betydelse:

- Auto: pumpen styrs av plc-programmet.
- Manuell: pumpen styrs av operatören från objektsbilden.
- Lokal: pumpen styrs från en pulpet.

I Mode objektet kan man konfigurera vilka moder som är aktuella för den här pumpen.

12.1 En Component fallstudie

I det här avsnittet ska vi programmera en reglering av nivån i en vattentank, mha komponenter och aggregat.

Process

Processen vi ska styra ser vi i figur 'Nivåreglering'. Vattnet pumpas från en reservoar till en tank. Vår uppgift är att reglera nivån i tanken. Till hjälp har vi en nivå-givare och en reglerventil. I systemet finns även en returledning med en magnetventil som alltid är öppen under regleringen. Vi noterar att tanken är 1 meter hög, vilket kommer att återspeglas i diverse olika min och max värden vid konfigureringen.

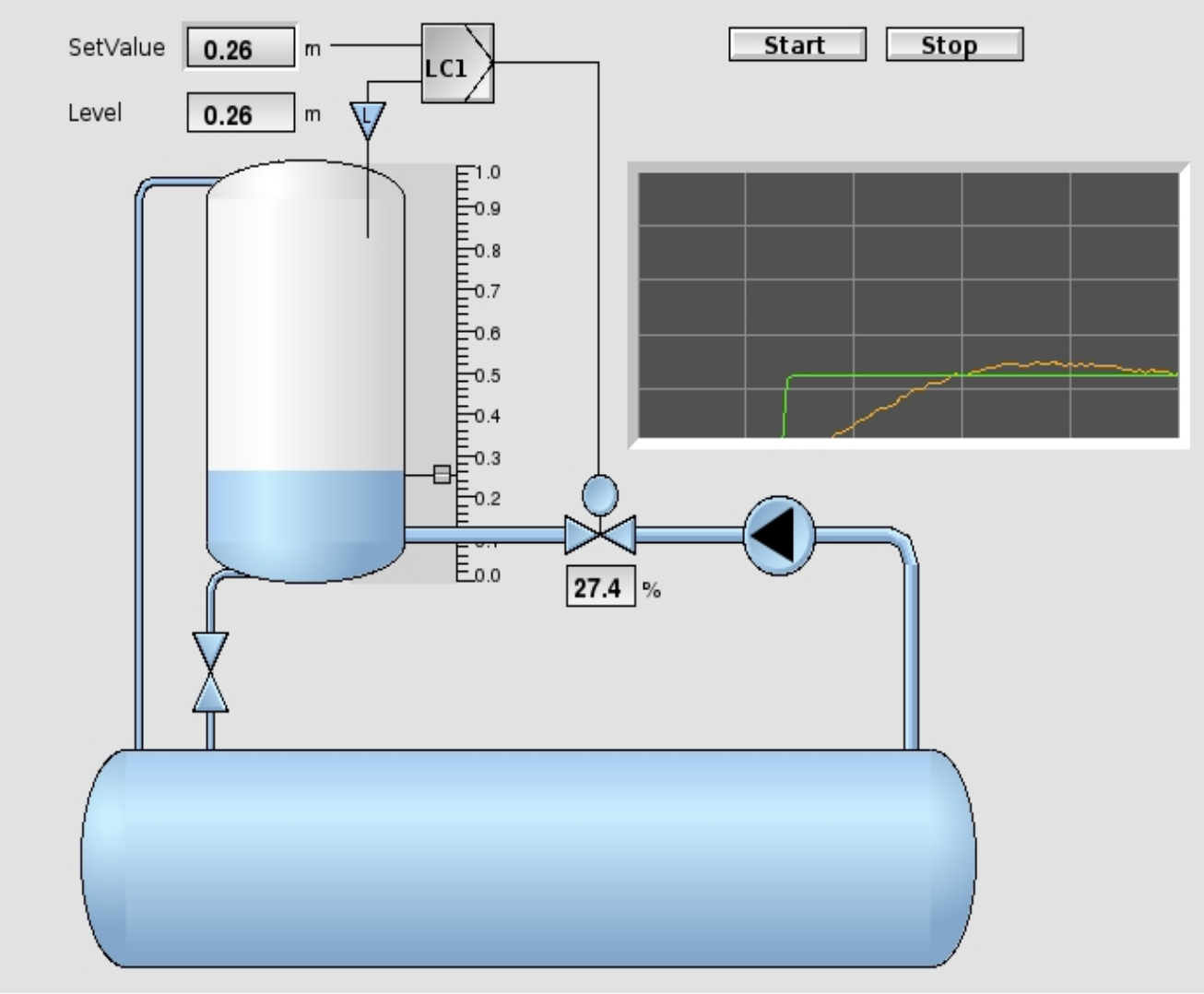


Fig Nivågreglering

Vi kan identifiera följande komponenter:
- en kontaktorstyrd pump, med effektbrytare, kontaktor, motorskydd och säkerhetsbrytare.
- en reglerventil med två gränslägen för öppen och stängd.
- en nivågivare.
- en magnetventil med två gränslägen för öppen och stängd ventil.

Följande komponentobjekt motsvarar komponenterna i anläggningen:

- Pump	BasePumpAggr
- Reglerventil	BaseCValve
- Nivågivare	BaseLevelSensor
- Magnetventil	BaseMValve

Konfigurering i planthierarkin

Komponenterna läggs under hierakin LevelControl. Under detta lägger vi ett PlcPgm 'Plc' som ska innehålla koden för styrningen, ett PlcPgm 'Simulate' som ska innehålla den kod som behövs för att vi ska kunna simulera och testa programmet. Vi skapar även tre Dv objekt för starta, stoppa och återställa reglerprogrammet (Start, Stop och Reset).

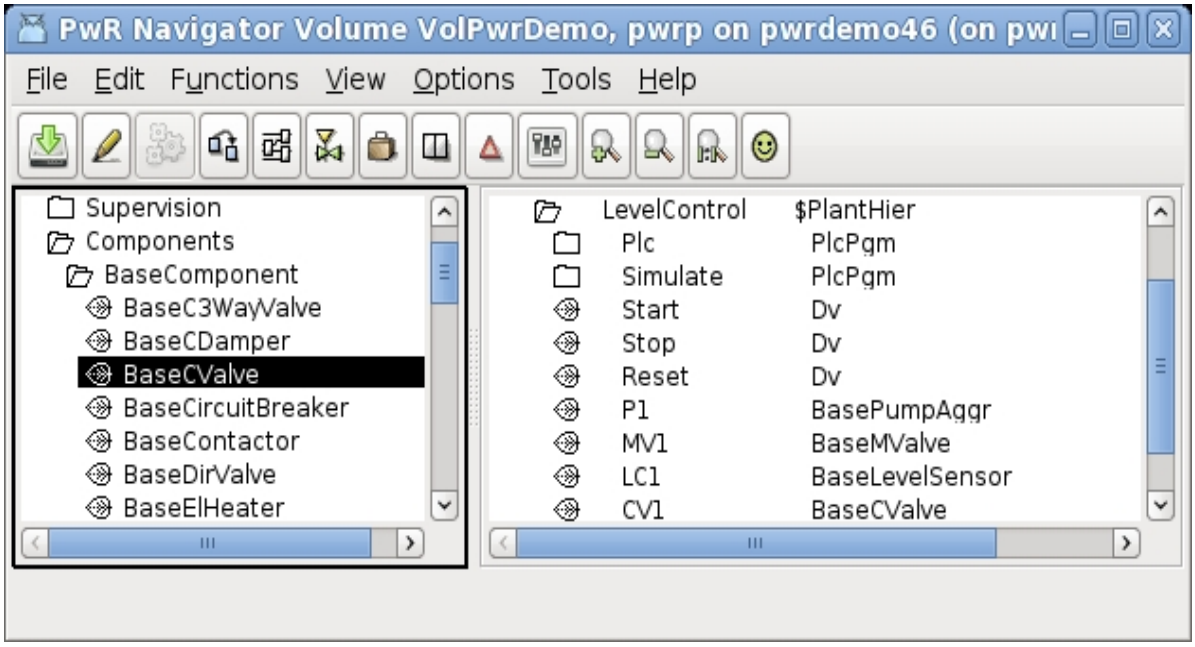


Fig Plant configuration

Pump

Pumpstyrningen består av
- en effektbrytare med en Di.
- en kontaktor med en Do för order, och en Di för kontaktorsvar.
- ett motorskydd med en Di.
- en säkerhetsbrytare med en Di.
- en motor utan några signaler.

För pumpen skapas ett BasePumpAggr objekt men namet P1.

Det ConfigureComponent alternativ som motsvara konstruktionen är CbCoOrSs (CircuitBreaker, Contactor, Order, SafetySwitch) och vi väljer ut P1 och aktiverar detta alternativ i ConfigureComponent från popupmenyn.

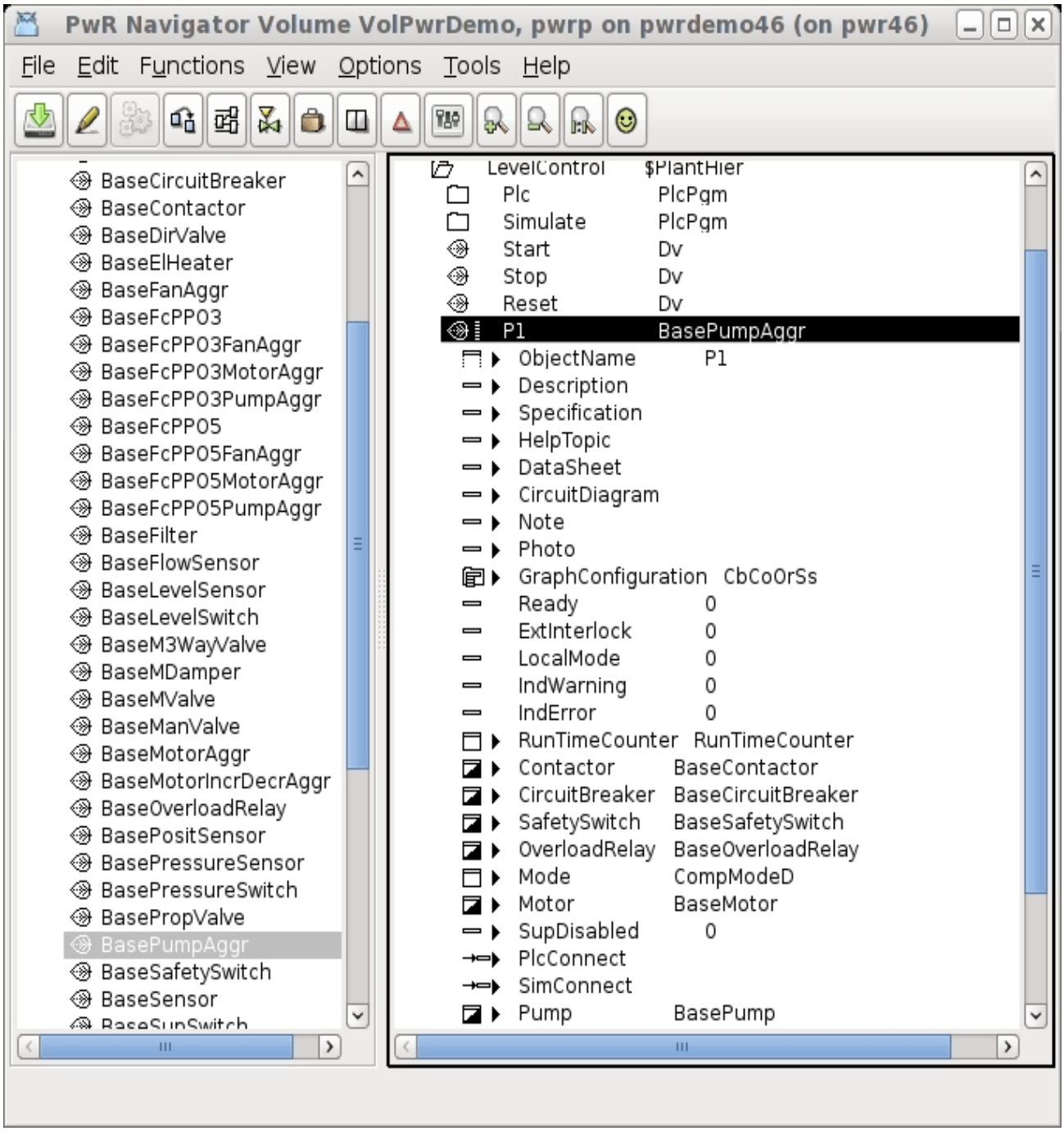


Fig Pump configuration

Komponenterna Contactor och Motor har egna konfigurationer så även för dessa måste vi aktivera ConfigureComponent. Vi väljer ut Contactor och väljer ConfigureComponent/OrderFeedback eftersom vi både har en signal for order och en för kontaktorsvar. För Motor objektet väljer vi ConfigureComponent/NoTempSwichOrSensor eftersom motorn saknar signaler.

Alla signaler som finns i pumpaggregatet ska kopplas till kanalobjekt i nodhierarkin. Vi letar upp följande signalobjekt och kopplar dem lämpliga kanaler:

P1.Contactor.Order	Do order till kontaktorn.
P1.Contactor.Feedback	Di kontaktorsvar.
P1.CircuitBreaker.NotTripped	Di effektbrytare ej utlöst.
P1.SafetySwitch.On	Di säkerhetsbrytare till.

P1.OverloadRelay.Overload	Di överlastningsrelä utlöst.

Reglerventil
Reglerventilen har ett ställdon som styrs med en analog utsignal, och gränslägesgivare för öppen och stängd ventil.

Vi skapar ett BaseCValve objekt med namnet CV1. ConfigureComponent alternativet som passar in på vår konstruktion är TwoSwitches.

Följande signaler kopplar vi till lämpliga kanaler i nodhierarkin:

CV1.Actuator.Order	Ao för styrsignal.
CV1.Actuator.SwitchOpen	Di för gränsläge öppen.
CV1.Actuator.SwitchClosed	Di för gränsläge stängd.

Nivågivare
För nivågivaren skapar vi ett BaseLevelSensor objekt men namnet LC1. Denna har inte någon ConfigureComponent metod, men det finns en del annat att konfigurera.

Givarobjektet har larmgränser för HögHög, Hög, Låg och LågLåg och dessa sätts i attributen LC1.LimitHH.Limit, LC1.LimitH.Limit, LC1.LimitL.Limit och LC1.LimitLL.Limit. Vi sätter även övre gränsen för visning av värdet i LC1.Value.PresMaxLimit till 1. Detta påverkar området för stapeldiagram och trendkurvor.

Magnetventil
Magnetventilen styrs av en digital utsignal och har feeback i form av digitala insignaler för gränsläge stängd och gränsläge öppen.

För magnetventilen skapar vi ett BaseMValve objekt med nament MV1.

I ConfigureComponent väljer vi TwoSwitches som motsvarar den aktuella konfigurationen med de båda gränslägena.

Signaler som ska kopplas till kanaler i noderhierarkin är:

MV1.Order	Do för order att öppna ventilen.
MV1.SwitchOpen	Di för gränsläge öppen.
MV1.SwitchClosed	Di för gränsläge stängd.

Plcprogram

Nästa steg är att skriva plcprogrammet för nivåstyrningen. Här ska funktionsobjekten för komponenterna ingå:

- BaseMotorAggrFo för pumpen P1.
- BaseCValveFo för reglerventilen CV1.
- BaseMValveFo för magnetventilen MV1.
- BaseSensorFo för nivågivaren LC1.

Vi lägger ut funktionsobjekten och kopplar dem till respektive huvudobjekt, genom att välja ut huvudobjektet och aktivera Connect i popupmenyn för funktionsobjektet.

Vi kommer även att använda en PID regulator för att styra nivån i tanken. Regulatorn har värdet från nivågivaren som ärvärde, och lägger ut utsignalen till reglerventilen, som då kommer att anpassa inflödet i tanket så att rätt nivå uppnås. Regulatorn skapas med

funktionsobjekten Mode och PID.

Programmet byggs upp kring en Grafcet sekvens med fyra steg. Se Fig Plc program

1. Initialsteget IS0 är viloläget när pumpen är avstängd och alla ventiler stängda.

2. När Start Dv'n sätts från en knapp i operatörsbilden, aktiveras steg S0. Här
   startas pumpen genom att sordern Ord0 kopplas till startingången på funktionsobjektet
   för pumpen P1. När pumpen har startat sätts On utgången på pumpobjektet, och
   aktiviteten flyttas till nästa steg S1. Observera att sordern Ord0 fortsätter att
   vara aktiv, dvs pumpen är igång, tills man gör reset i steg S2.

   Från fel-utgången Err på pumpobjektet sätter vi Reset Dv'n. Reset är inlagd som
   ResetObject i PlcPgm objektet, vilket gör att alla sekvenser i PlcPgm'et kommer
   att återställas när Reset sätts, vilket medför att pumpen och regleringen stängs av
   och sekvensen återgår till vilotillståndet i IS0.

3. S1 är driftsteget, där regulatorn är till, och reglerar nivån i tanken.
   Så länge S1 är inaktivt, sätts force-ingången på mode objektet och regulatorn har
   0 som utsignal. När steget blir aktivt, hämtar regulatorn ärvärdet från utgången från
   givarobjektet LC1, börvärdet sätt i modeobjektet LC1_Mode.SetVal från operatörsbilden,
   och utsignalen är kopplad till reglerventilens order-ingång.
   Ordern Ord1 är även kopplade till magnetventilens order-ingång och öppnar denna.

4. Steg S1 är aktivt tills Stop Dv'n sätt från en knapp i operatörsbilden. När steget
   lämnas tvångstyrs regulatorn igen och reglerventilen stänger. Även magnetventilen stängs.
   Steget S2 passeras som hastigast och gör reset på sordern Ord0, dvs stoppar pumpen.
   Därefter återgår sevensen till viloläget IS0.

Mode objektet LC1_Mode och regulatorn LC1_PID kräver lite ytterligare konfigurering.
I modeobjektet sätts
- OpMode = Auto så att regulatorn startar i auto-mod.
- MaxSet = 1, max börvärde är höjden på tanken, 1m.
- SetMaxShow = 1, också höjden på tanken.
- SetEngUnit = m, meter.
- PidObjDid = LevelControl-Plc-W-LC1_PID, namnet på PID objektet.

I PID objektet sätts
- PidAlg = PID
- Gain = 100
- IntTime = 10
- MaxGain = 200
- SetMaxShow = 1
- SetEngUnit = m
- ModeObjDid = LevelControl-Plc-W-LC1_Mode, namet på mode objektet.

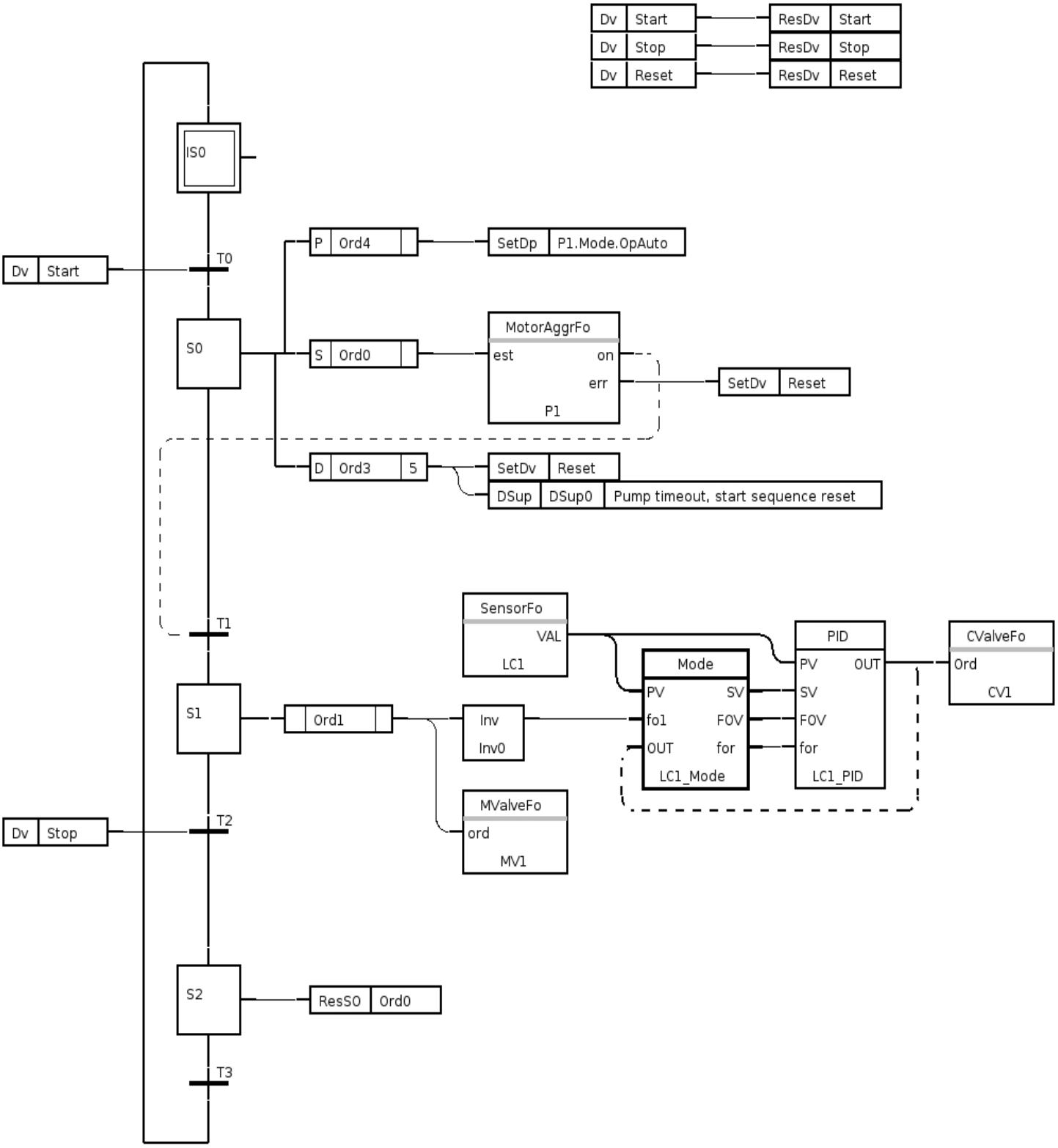


Fig Plc program

Simulerings program

Simuleringsprogrammet är till för att kunna testa alla funktioner i programmet, larmhantering och operatörbilderna före idrifttagningen. Man kan även använda det för utbildning och

demonstration.

Simulerings-koden läggs i ett separat program 'Simulate' , som inte ska exekvera i driftsystemet. I detta läggs simuleringsobjekt för komponenterna:

- BaseMotorAggrSim för pumpen P1.
- BaseCValveSim för reglerventilen CV1.
- BaseValveSim för magnetventilen MV1.
- BaseSensorSim för nivågivaren LC1.

Funktionsobjekten kopplas till huvudobjekten, genom att välja ut huvudobjektet och aktivera Connect i popmenyn för funktionsobjeket.

Simuleringsobjekten för pumpen, reglerventilen och magnetventilen har inte några in eller utgångar utan arbetar uteslutande mot data i huvudobjektet, där de läser utgångssignaler och sätter lämpliga värden på ingångssignaler. Simuleringsobjekten har även en objektsbild, där man kan påverka simuleringen och framkalla olika felfall för att se att felen hanteras på ett kontrollerat sätt och att operatören informeras via processbilder och larmhantering.

Simuleringsobjektet för nivågivaren har däremot en ingång, och här måste vi beräkna en simulerad nivå i tanken. En ändring i nivån bestäms av infödet - utflödet dividerat med tankens area. Vi antar att inflödet är proportionellt mot order utgången till reglerventilen (CV1.Actuator.Order), och drar ifrån utflöde genom magnetventilen när denna är öppen. Inflödet accumuleras i CArithm'ens OA1 utgång, som skickas vidare till simulerings objektet för nivågivaren LC1. I LC1 adderas även lite brus till signalen för att få ett mer realistiskt utseende, genom att LC1.RandomCurve sätts till 1, och LC1.RandomAmplitude till 0.1.

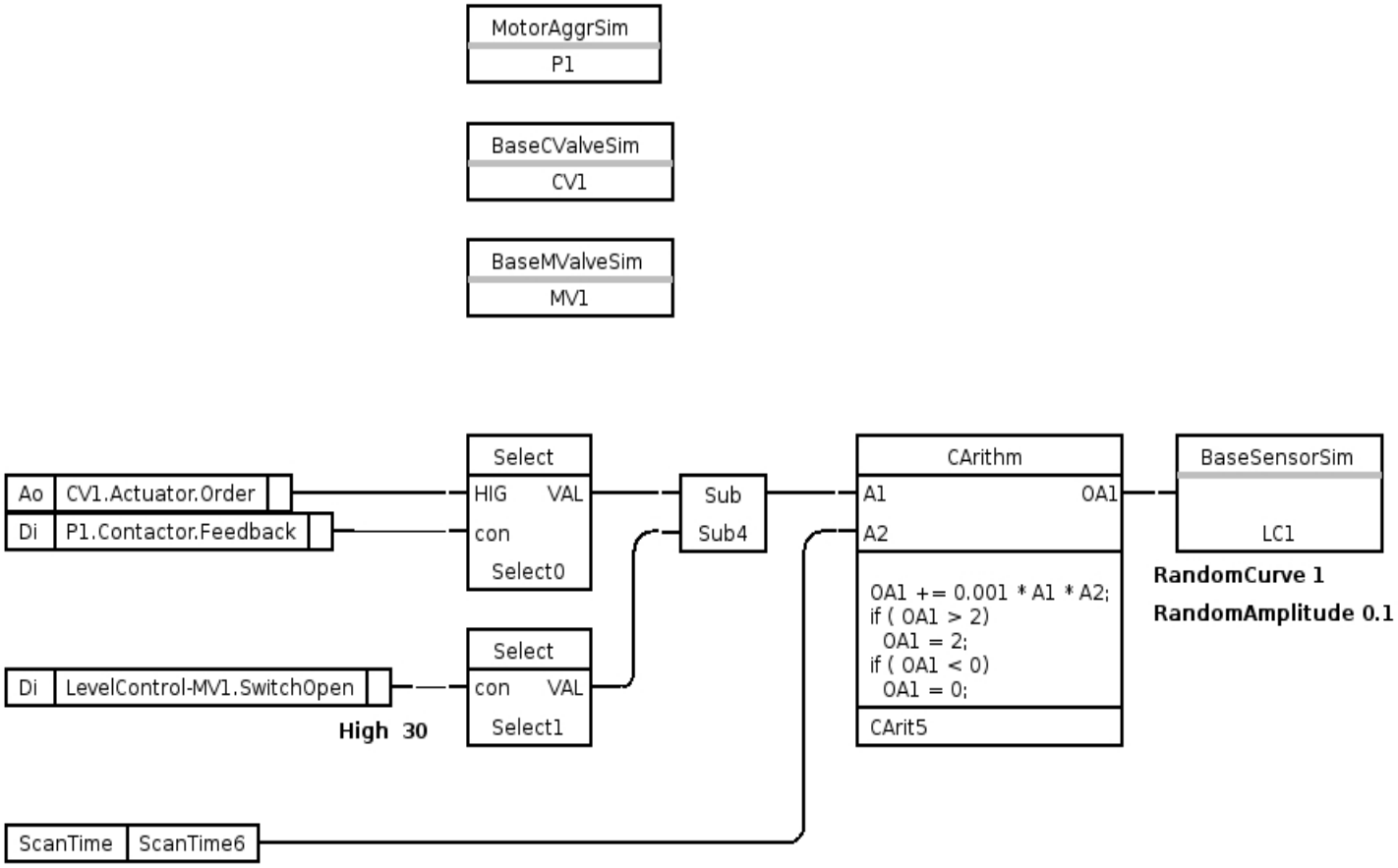


Fig Simuleringsprogram

Processbild

Operatörsbilden för nivåstyrningen ritas i Ge, och här används de grafiska symbolerna för komponenterna som finns i Ge paletten:

- Component/BaseComponent/PumpAggr för pumpen.
- Component/BaseComponent/CValve för reglerventilen.
- Component/BaseComponent/MValve för magnetventilen.
- Component/BaseComponent/LevelSensor för nivågivaren.

Symbolerna har en dynamiken HostObject, som innebär att de har en förprogrammerad dynamik som är kopplad till olika attribut i objektet. Man behöver endast lägga in objektsnamnet på huvudobjektet HostObject.Object, eller koppla dem genom välja ut huvudobjektet och klicka med Dubbelklick Ctrl/MB1 på symbolen.

I default dynamiken ingår inte att objektsbilden öppnas när att klicka på symbolen, vi adderar denna funktion genom att öppna objektattribut editorn för symbolen och lägga till OpenGraph i action (om man inte anger något OpenGraph.GraphObject objekt öppnas objektbilden).

Vi konstruerar också en tank, mha en fyrkant och två halva ellipser, som vi sätter fill och gradient på och även grupperar. På gruppen sätts dynamik FillLevel och FillLevel.Attribute kopplas till värdet för nivågivaren, dvs LevelControl-LC1.Value.ActualValue. Min och max värdet för FillLevel anges till 0 och 1.

Till vänster om tanken läggs en slider för att enkelt kunna ställa in börvärdet för nivån. Den består av en Slider/SliderBackground3 och en Slider/Slider3. Slidern kopplas till börvärdet i mode objektet LC1_Mode, dvs LevelControl-Plc-W-LC1_Mode.SetVal. Min och maxvärde för slidern sätts till 0 och 1.

För börvärdet skapas också ett inmatningsfält 'SetValue' som kopplas till samma värde som slidern ovan. Ärvärdet för nivån visas i value fältet 'Level' och kopplas till värdet för nivågivaren.

Regulatorsymbolen hämtas från Process/PID_Controller i paletten. Den har ingen dynamik som default, men vi vill att objektsbilden för mode objektet ska öppnas när man klickar på den, så vi lägger in Action Command med kommandot

```
open graph /class/instance=LevelControl-Plc-W-LC1_Mode
```

Tryckknapparna för Start och Stop är av typen Buttons/SmallButton. SmallButton har ToggleDig som default action, men vi vill ha SetDig istället eftersom Start resp Stop signalerna återställs av plcprogrammet. Vi tar bort Inherit i Actions för att undvika ToggleDig och lägger dit SetDig i stället, och kopplar till Start resp Stop Dv'n.

Trendkurvan hämtas från Analog/Trend i paletten. I denna ska ärvärdet och börvärdet visas, dvs
- Trend.Attribute1 sätts till LevelControl-LC1.Value.ActualValue (värdet för nivågivaren).
- Trend.Attribute2 sätts till LevelControl-Plc-W-LC1_Mode.SetVal (börvärdet i mode objektet).

Slutligen ritas lite rör och linjer och bilden är klar.
Vi går även in i File/Graph attributes och lägger in koordinaterna för övre vänstra och under högra hörnen i x0,y0 och x1,y1. DoubleBuffered sätt till ett och MB3Action till PopupMenu.

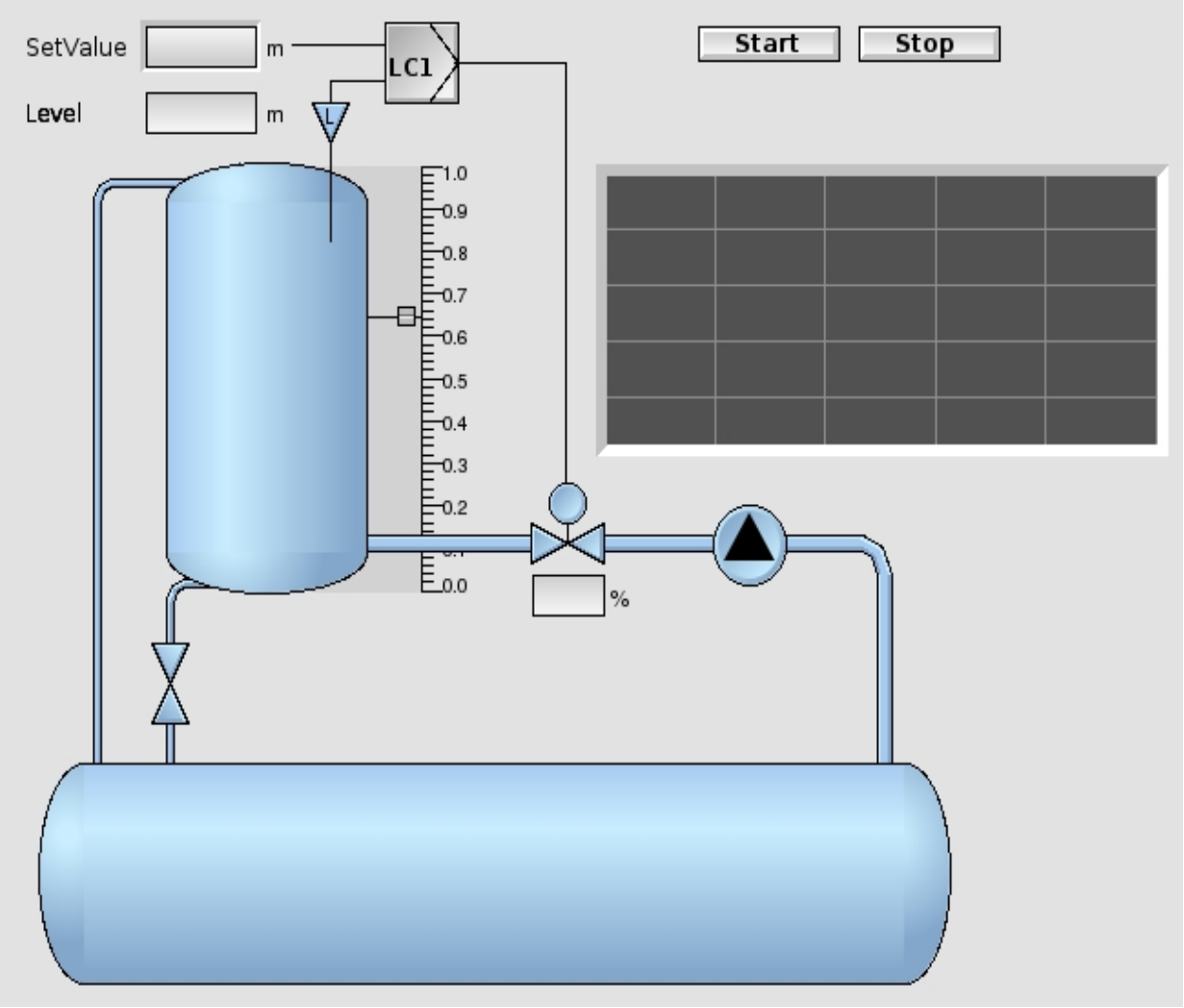


Fig Graphic editor

Simulering

För att se resultatet av vår programmering startar vi upp simuleringen.

När vi öppnar bilden är de båda ventilerna färgade vita, vilket markerar att de är stängda. Pumpen är inte startad, vilket markeras med grå färg och att trekanten i pumpsymbolen inte pekar i strömningsriktningen. Tanken är vitfärgad, dvs tom, och nivågivaren blinkar ilsket rött eftersom nivån ligger under LowLow nivån i nivågivarobjektet.

Genom att trycka på startknappen lämnar vi vilosteget i Grafcet sekvensen, och aktiverar steget som startar pumpen (S0). När pumpen har startats färgas den blå och trekanten pekar i strömningens riktning. I det här steget öppnas även magnetventilen som färgas blå. Så snart pumpen har startat övergår sekvensen till driftsteget S1.

Vi lägger in ett börvärde med slidern på ca 0.3 och förhoppningsvis börjar regulatorn att arbeta. Det är möjligt att det behövs lite intrimning av regulatorn, och regulatorbilden får vi upp genom att klicka på regulator symbolen, som öppnar objektsbilden för Mode objektet. I denna klickar vi på PID knappen som öppnar objektbilden för PID objektet. Här kan vi trimma på förstärkning (Kp) och integrationstid (Ti).

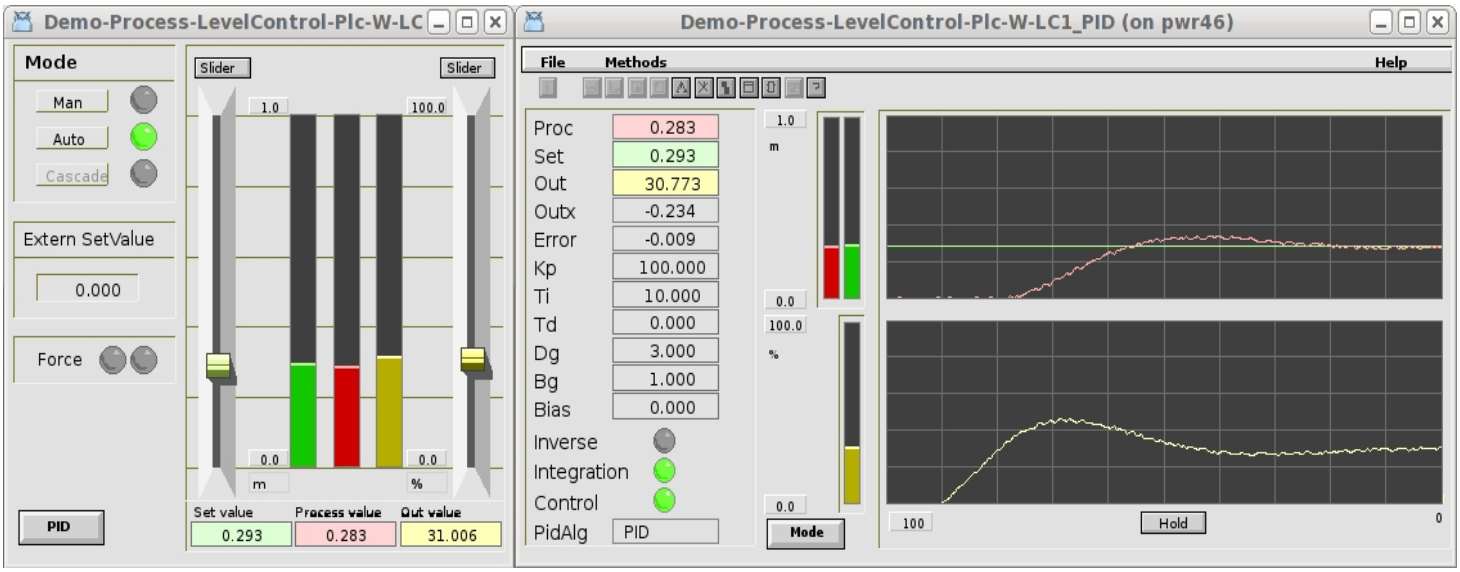


Fig Object graph mode and PID object.

Låt oss titta lite på vad vi kan göra med komponenterna i bilden.

Nivågivare

Om vi högerklickar på nivågivaren får vi upp en popupmeny med de metoder som är definierade för givaren. Med OpenPlc öppnar vi plc trace för komponentens funktionsobjekt, med RtNavigator letas objektet upp i navigatören, med Trend visas en trendkurva för nivån, med OpenGraph öppnas objektsbilden. Objektsbilden kan även öppnas genom att klicka på symbolen.

Den övre delen av objektsbilden för komponenter och aggregat har ett likartat utseende. Där finns en meny där man under 'Methods' kan aktivera metoderna för komponenten. Under 'Signals' kan man se vilka signaler som finns i komponenten och öppna objektbilden för dem. För aggregat kan man även se ingående komponenter och öppna objektsbilden för dessa under 'Components'. Det finns också en vertygspanel med knappar för metoderna, och två textfält som visar Description och Specification för komponenten. På den nedersta raden i bilden visas ett Note meddelande om något sådant är inlagt (detta läggs in med Note metoden).

I övrigt visas i nivågivarbilden nivån, i sifferform och i stapelform, och de larmnivåer som finns konfigurerade. Larmnivåerna kan ställas in med sliders och kan även aktiveras eller inaktiveras med checkboxar.

Längst upp till höger i bilden finns en knapp markerad med 'S'. Den är bara synlig om man kör simulerat (IOSimulFlag i IOHandler objektet är satt) och med den öppnar man simuleringsbilden. Från simuleringsbilden kan man påverka den simulerade signalen. Vi har redan konfigurerat Random med amplituden 0.01, man kan även superponera en sinuskurva eller sågtandskurva på signalen.

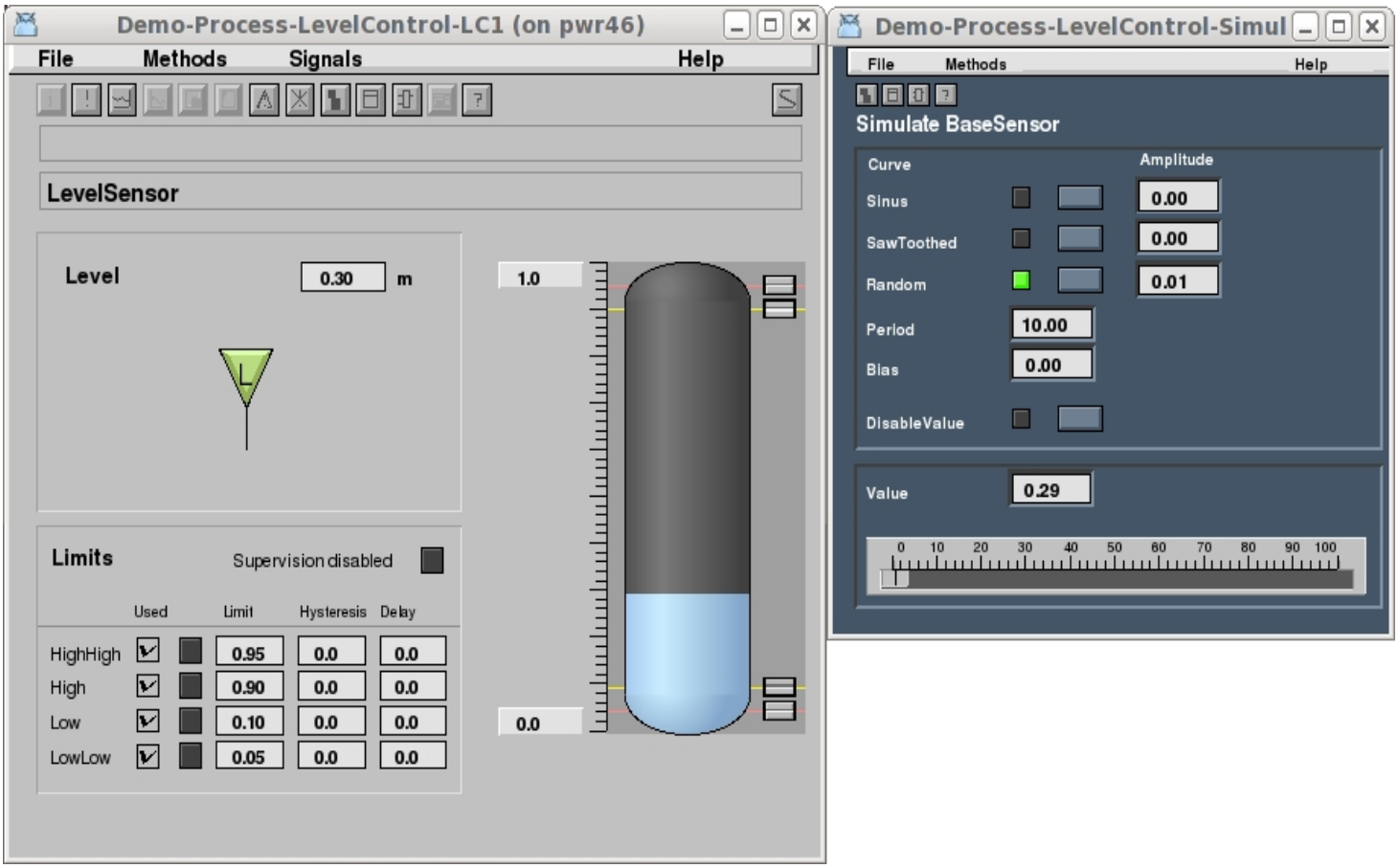


Fig Object graph for the levelsensor and simulate graph

Reglerventil

Objektsbilden för reglerventilen har indikatorer för gränslägena och visar ordern som läggs ut till ventilen i stapelform.

Vi kan ta över ventilen i manuell mod, dvs istället för att regulatorns utsignal läggs ut till ventilen, ställs ventilläget in med slidern 'Manual'. Regleringen är nu satt ur spel och vattenflödet styrs mha slidern.

Från simuleringsbilden kan man t ex påverka simuleringen av gränslägena. Om Order är 0 och inte gränsläge stängd är påverkad får man ett gränslägesfel med larm och rödblinkande symbol. Vid simulering ser simuleringsobjektet till att ställa in gränslägena rätt, men detta kan överridas från simuleringsbilden. Genom att trycka på 'Manual Control' styrs gränsläget från bilden istället, och genom att ta ner gränslägessignalen kan man testa att gränslägesövervakningen fungerar.

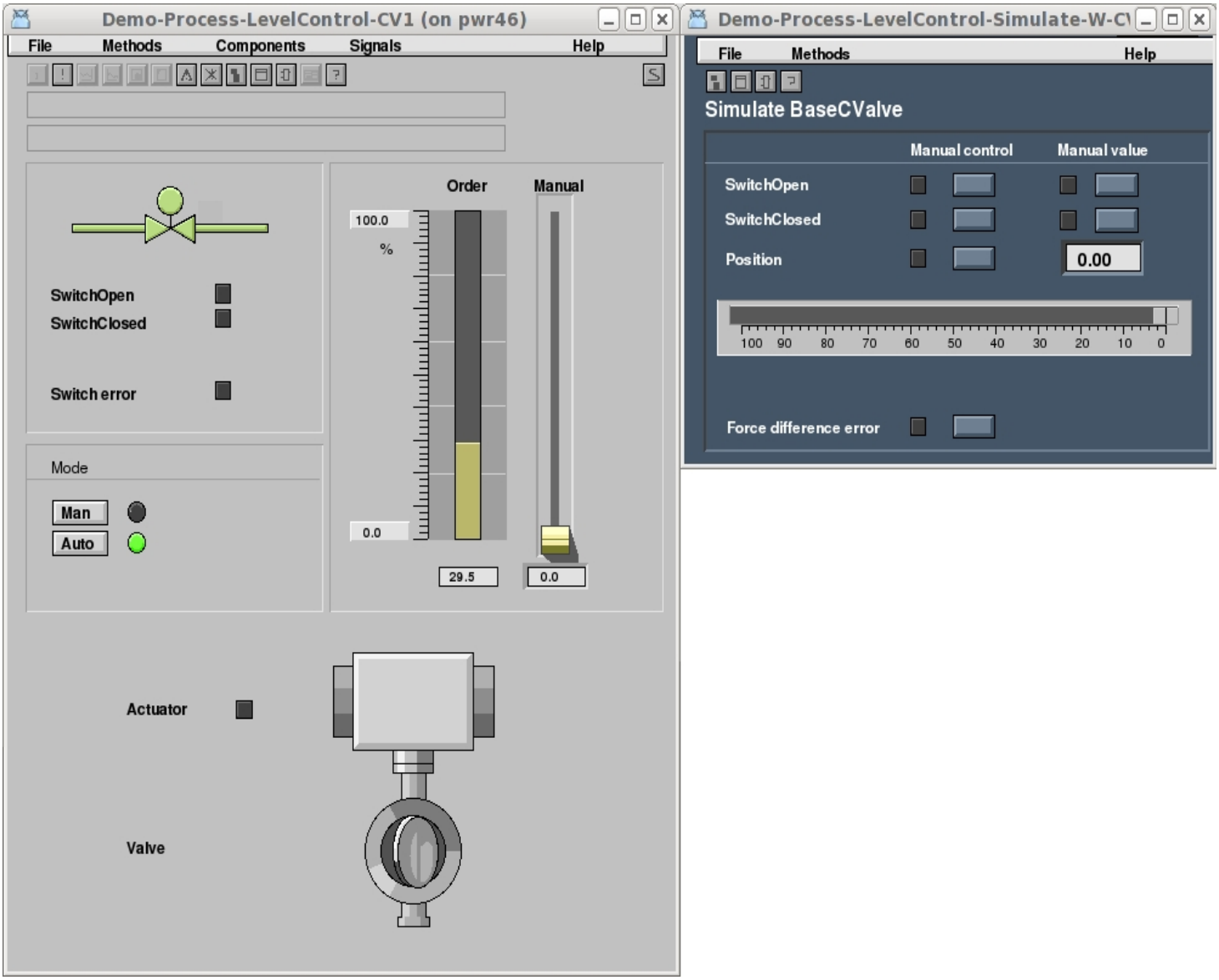


Fig Object graph for the controlvalve and simulate graph

Magnetventil

Objektbilden för magnetventilen visar gränslägen och order signal med indikatorer. Ventilen tas över i manuel mode genom att klicka på Man knappen, och kan nu manövreras med Open och Close knapparna.

I simuleringsbilden kan man liksom för reglerventilen påverka simuleringen av gränslägena och framkalla larm för gränslägesfel.

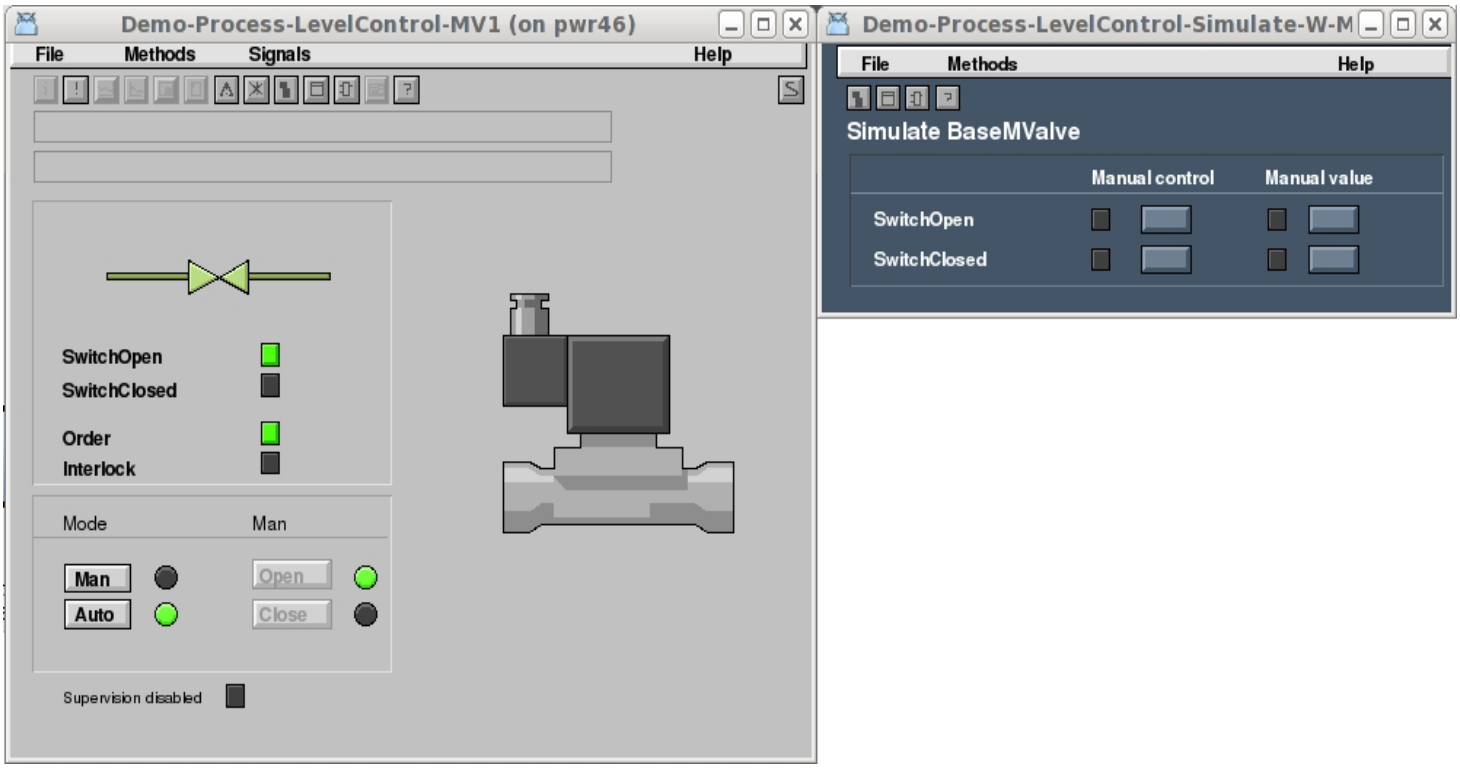


Fig Object graph for the solenoidvalve and simulate graph

Pump

Objektsbilden för pumpen visar en schematisk skiss över de ingående komponenterna, och även en status indikator för varje komponent. Genom att klicka på en komponent öppnar man objektsbilden för denna.

Med 'Man' knappen kan man ta över pumpen i manuell mod och starta och stoppa pumpen från Start och Stop knapparna i bilden.

I simuleringsbilden kan olika händelser man simuleras.
- 'SafetySwich on' simulerar att någon sätter på säkerhetsbrytaren. Det medför att pumpen stängs av, pumpsymbolen färgas gul, och att Err utgången på funktionsobjektet sätts. Denna har vi kopplat till Reset Dv'n som återställer sekvensen och stänger av regleringen.
- 'CircuitBreaker tripped' simulerar att effektbrytaren löser ut.
- 'Contacor feeback lost' simulerar att kontaktorsvaret tappas.
- 'OverloadRelay tripped' simulerar att överlastningsrelät löser.

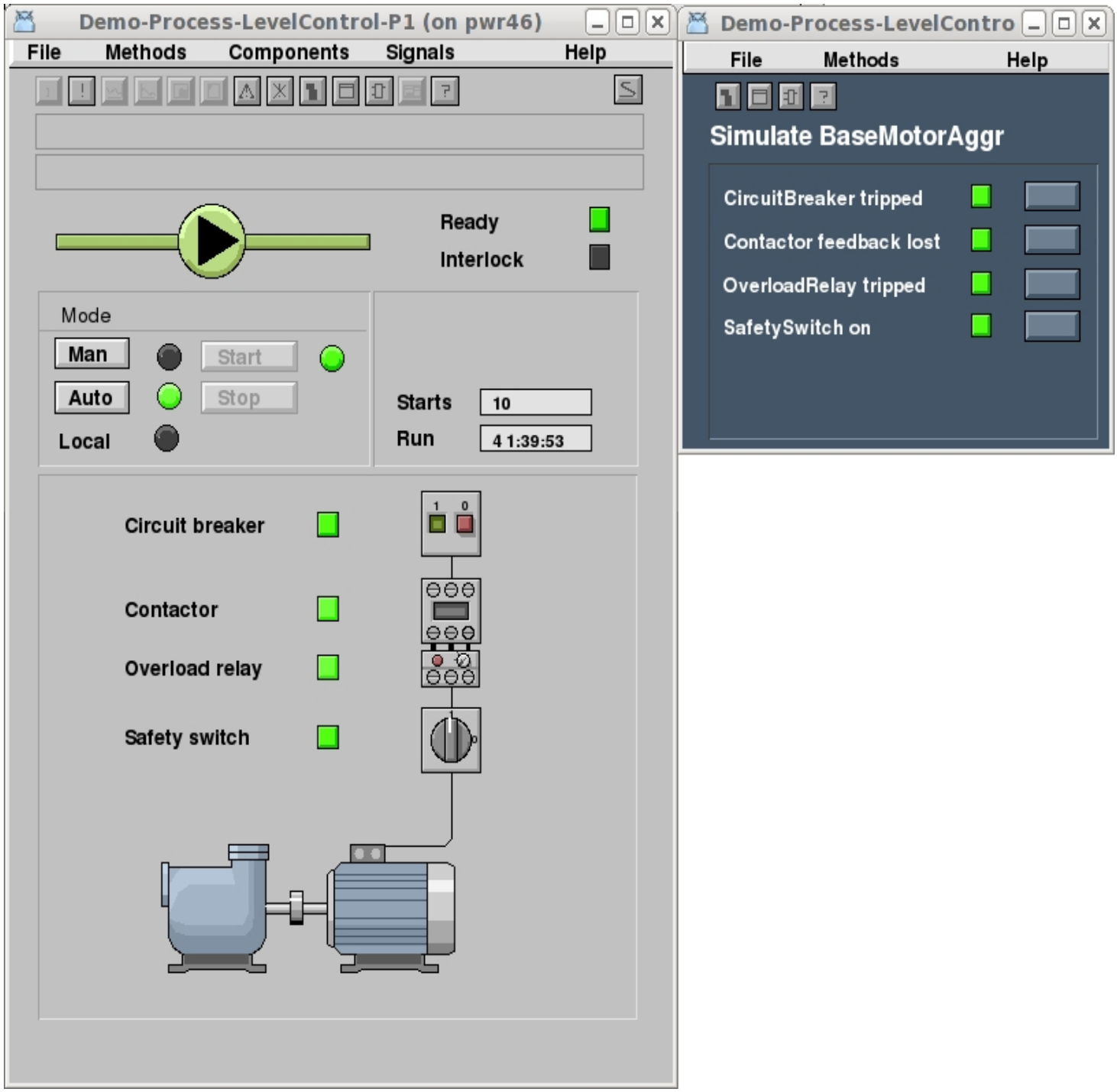


Fig Object graph for the pump and simulate graph

13 Larm och händelser

Larmhanteringen i ProviewR presenterar larm, informationsmedelanden och händelser i två listor, larmlistan och händelselistan. Larmlistan innehåller rådande eller okvitterade larm och info-meddelanden, och ger operatörer och underhållspersonal information om det nuvarande läget i processen. I händelselistan loggas händelser som t ex larm som aktiveras, återgår och kvitteras, så att man kan få en bild av vad som har hänt i systemet.

Larm och händelser genereras av övervakningsobjekt. De vanligaste är DSup- och ASup-objekt som övervakar digitala respektive analoga signaler. Övervakningsobjekten skannas av eventmonitorn som skickar ut meddelanden om larm och händelser till olika utenheter, t ex larm och händelselistan i xtt.

13.1 Larm

Ett larm är en tillståndsförändring i processen eller styrsystemet som kräver operatörens uppmärksamhet. Larmet presenteras på en larmlista, ofta med en ljudsignal, och operatören måste kvittera larmet innan ljudsignalen upphör och larmet tas bort från larmlistan. Även larmtillståndet måste återgå innan larmet tas bort.

Ett larm består av tre händelser
- larmet går till.
- larmet återgår från larmtillstånd till normalt tillstånd.
- larmet kvitteras av en operatör.

Prioritet
Ett larms prioritet kan vara A, B, C eller D, där A har högst prioritet. Olika prioriteter har olika färgmarkering. A-larm markeras med rött, B-larm med gult, C-larm med blått och D-larm med violett.

Typ
Det finns 7 olika typer av larm som kan användas för att addressera larm till olika kategorier av mottagare

Alarm	Vanligt larm för driftsoperatörer.
MaintenanceAlarm	Larm för underhållspersonal.
SystemAlarm	Larm för systemansvarig.
UserAlarm1-4	Användardefinierade larmtyper.

Larmtypen matchas mot EventSelectType i OpPlace-objektet, och i larmlistan visas endast de valda larmtyperna.

13.2 Info-meddelanden

Info-meddelanden liknar larm men har inte någon prioritet. Man kan även konfigurera dem så att de inte visas i larmlistan, utan enbart i händelselistan.

Typ

Det finns två typer av Info-meddelande, Info som markeras med vitt, och InfoSuccess som markeras med grönt.

13.3 Händelser

Händelser är en förändring i processen eller styrsystemet som loggas i en händelselista. De används för att se vad som har hänt i processen, t ex för att kunna dra slutsatser om händelseförloppet om något har gått fel.

En typ av händelser är larm som går till, återgår och kvitteras, men även andra händelser kan loggas i händelselistan.

13.4 Övervakningsobjekt

Larm och händelser konfigureras med övervakningsobjekt, främst DSup- och ASup-objekt. Ett DSup-objekt övervakar en digital signal, och genererar ett larm eller en händelse när den digitala signalen indikerar larmtillstånd. Ett ASup-objekt övervakar en analog signal, och generar larm eller händelser om det analoga värde passerar en larmgräns. Andra övervakningsobjekt är CycleSup som övervakar cykliska processer, NodeLinkSup som övervakar länkar till andra noder och SystemSup som övervakar funktioner i styrsystemet.

DSup och ASup

DSup och ASup är de vanligaste övervakningsobjekten och övervakar digitala resp analoga signaler. De finns även i varianterna CompDSup och CompASup som används i komponenter och aggregat.

DSup och ASup objekten kan antingen läggas i plc koden, och kopplas till det digitala resp analoga värde som ska övervakas, eller läggas i planthierakin, t ex under en signal. Om Sup-objektet läggs under en signal kommer ActualValue att övervakas genom att detta automatiskt läggs in i Attribute i Sup-objektet. Om Sup-objeket läggs under något annat objekt måste Attribute fyllas i manuellt för att markera vilket attribut i objektet som ska övervakas.

DSup och ASup kan generera larm, info-meddelanden och händelser. Konfigureringen av attributen EventType, EventPriority och EventFlags avgör vilken typ larm och händelser som genereras.

EventType

EventType konfigurerar vilken typ av larm eller info-meddelanden som genereras. De olika larmtyperna är Alarm, MaintenanceAlarm, SystemAlarm och UserAlarm1-4. Info-meddelanden kan vara Info eller InfoSuccess.

EventPriority

EventPriority konfigurerar prioriteten för larm. Prioriteten kan vara A, B, C eller D. För info-meddelanden har prioriteten ingen betydelse.

EventFlags

EventFlags är en bitmask med flera val som avgör vilka händelser som genereras och vart de skickas. Se Objekthandboken för mer info.

Olika typer av larm och händelser

A larm med beep

EventType	Alarm
EventPriority	A
EventFlags	Return, Ack, Bell

B larm med beep

EventType	Alarm
EventPriority	B
EventFlags	Return, Ack, Bell

Info-meddelande som visas i larmlistan

EventType	Info
EventPriority	-
EventFlags	Ack, Bell, InfoWindow, Returned

InfoSuccess-meddelande som visas i larmlistan

EventType	InfoSuccess
EventPriority	-
EventFlags	Ack, Bell, InfoWindow, Returned

Info-meddelande som enbart visas händelselistan

EventType	Info
EventPriority	-
EventFlags	Returned

NodeLinkSup

Ett NodeLinkSup objekt övervakar nätverkslänken till en annan nod, och om länken går ner generaras ett larm.

SystemSup

I MessageHandler-objektet ligger en vektor med SystemSup-objekt som övervakar funktioner i styrsystemet. Ett larm eller info-meddelande skickas vid följande tillfällen

- Någon nätverkslänk går ner.
- Vid systemstart.
- Vid omstart.
- Vid omstart av en utenhet för larmhanteringen.
- Systemet nödstoppar.
- När systemstatus indikerar varning eller fel.
- När felräknaren på en IO-enhet når soft eller hard limit.
- När kvota för larm har överskridits.

13.5 Eventmonitorn

Serverprocessen som hanterar larm och händelser kallas för event monitorn och har namnet rt_emon. Det konfigureras med ett MessageHandler objekt som läggs i nodhierakin.

I MessageHandler-objektet konfigureras bl a EventListSize som är antalet händelser som lagras internt i event monitorn, och EventLogSize anger storleken på event loggen, dvs händelser som lagras på disk.

Event monitorn skannar av alla sup-objekt och skickar ut information om larm och händelser till olika utenheter, t ex larm och händelselista. När en utenhet startas skickas alla händelser i den interna listan, och sedan skickas nya händelser allteftersom de inträffar. Det skickas även en status-trans med aktiva eller okvitterade larm så att larmlistan alltid ska vara korrekt uppdaterad.

13.6 Blockering av larm

Om en del av anläggningen är avstängd kan man blockera larmen ifrån den. Det görs från navigatorn genom att öppna popupmenyn för en hierarki och välja 'Blockera händelser'. Blockeringen görs med prioritet så att larm med den valda prioriteten och lägre prioritet blockeras. En lista på aktiva blockeringar kan öppnas från en knapp i operatörsfönstret, och härifrån kan även blockeringar tas bort.

13.7 Undertryckning av larm

Ett vanligt fel i larmsystem är att ett fel ger upphov till en kaskad av larm som gör det omöjlig för operatören att hitta grundorsaken. Det kan undvikas genom att man undertrycker larm som inte är releventa i en viss situation.

Undertryckning gör med funktionsobjektet SuppressSup. I objektet anges ett övervaknings-objekt, och när ingången blir hög blockeras larmet från objektet. Ingången kan t ex kopplas till Active-utgången på ett DSup-objekt som i exemplet nedan.

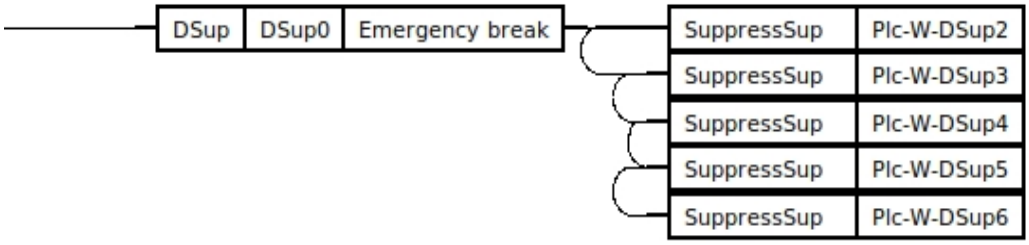


Fig Undertryckning av larm

13.8 Utenheter

13.8.1 Larm och händelselista i xtt

Larm och händelselistan i operatörsmiljön kan öppnas från knappar i operatörsfönstret, och även med xtt-kommandona

```
xtt> show alarmlist
xtt> show eventlist
```

I OpPlace-objektets EventSelectList anger man hierakier som larm och händelser ska visas ifrån.

Larm och händelselista i en XttMultiView

Det finns en funktion som skapar kopior, s k satelliter, av larm och händelselistorna. Dessa används för att visa larm eller händelselistan i en XttMultiView. Action.Type för den aktuella cellen sätts till AlarmList resp EventList.

AlarmView

En satellit av larmlistan kan även kombineras med en AlarmView, som kategoriserar larmen och lägger in dem under olika mappar. Under AlarmView-objektet läggs AlarmCategory-objekt, ett för varje mapp i listan. I AlarmCategory-objektet anges hierarkier, prioritet eller larmtyp för de larm som ska finnas under mappen. En mapp som innehåller aktiva eller okvitterade larm markeras med färg som visar prioriteten på larmen i mappen, och även antalet okvitterade larm under mappen visas.

Ett AlarmView-objekt ska anges i OpPlace-objektets AlarmView vektor.

En larmlist med AlarmView kan öppnas med xtt-kommandot

```
xtt> show alarmlist satellite /alarmview=
```

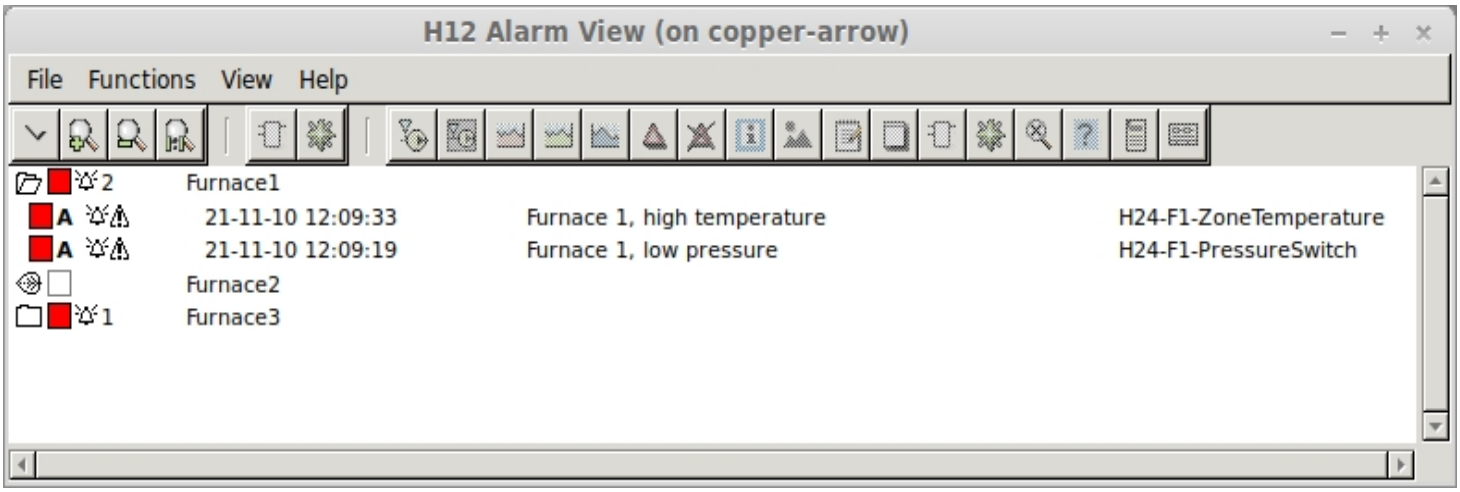


Fig Larmlista med AlarmView

Det kan även öppnas i en XttMultiView genom att ange AlarmView-objeket i Action[].Object[0] i XttMultiView objektet.

AlarmTable

Med AlarmTable-objektet kan man visa de senaste larmen in en operatörsbild. AlarmTable-objektet läggs under OpPlace-objektet och larmlistan för OpPlace-objektet kopieras till vektorer i AlarmTable-objektet. Det finns vektorer för EventText, EventName, Time, EventPrio mm, som kan visas upp i en tabell i operatörsbilden. Genom Member-attribute kan man göra ett urval av larm från specifika hierarkier, men är begränsad av OpPlace-objektets EventSelectList.

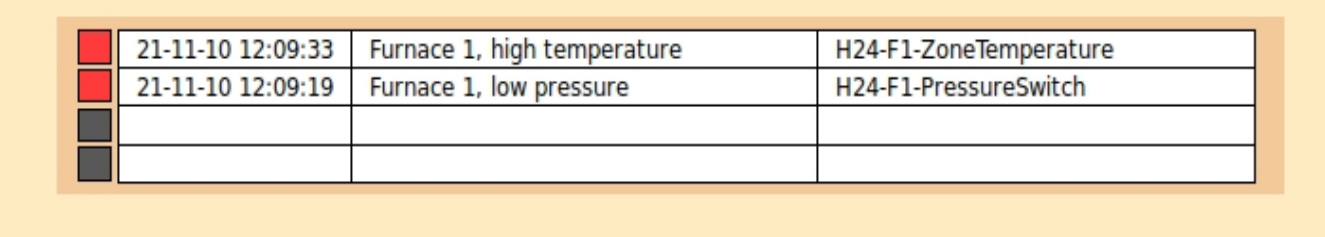


Fig Ett AlarmTable-objekt som visas i en Ge graf.

13.8.2 Larm och händelselista i web-gränssnittet

I operatörsplatsens meny finns knappar för att öppna larm och händelselista. Vilka larm som ska visas anges i WebSockerServer-objektets EventSelectList.

13.8.3 Eventloggen

I eventloggen lagras alla händelser i en databas. Det konfigureras genom att man anger storleken på händelseloggen i EventLogSize i MessageHandler-objektet. Från operatörsfönstret kan man öppna en dialog för att söka i eventloggen efter händelser.

Händelseloggen hanteras av serverprocessen rt_elog och ligger på $pwrp_db/rt_eventlog.db.

13.8.4 Historisk lagring av händelser

Händelser kan även lagras i historik-databasen. Konfigureringen sker med ett SevHistEvent-objekt där man anger bl a vilka larm som ska lagras och hur länge de ska lagras på servern.

Lagrade händelser kan visas av en Larm och Händelseanalysatorn.
Läs mer i Handbok i ProcessHistorik

14 Kommunikation

14.1 Intern kommunikation

Den interna kommunikationen i ProviewR skickar information om volymer, objekt, larm, händelser, historisk data mm mellan olika processer och noder. Det finns tre olika protokoll för larmhantering, näthantering och historisk lagring, som alla basera på Qcom.

Qcom

Qcom är en meddelande buss som skickar köade meddelanden mellan processer. Alla noder som ska kommunicera med varandra måste ligga på samma buss. Vilken buss man använder konfigureras i BusConfig objekt genom att ange buss-identiteten, ett tal mellan 1 och 999.

Man måste även konfigurera vilka noder som ska kommunicera med varandra. Med standard-inställningen kommera alla noder i samma projekt att kommunicera med varandra. Dessutom kan man ange noder i andra projekt med FriendNodeConfig objekt. Det kan vara process-stationer som ska monteras av operatörs-stationer eller historiska logg-servrar.

Om man har projekt med flera noder, och inte vill att alla ska kommunicera med varandra, kan man sätta QComAutoConnectDisable i BusConfig och använda FriendNodeConfig objekt för definiera länkar mellan noder i projektet.

Alla Qcom meddelanden sker med kvittens. Om kvittensen uteblir skickas meddelandet om med fördubblad timeout tid. Efter ett visst antal omsändningar utan svar betraktas länken som nere. Beroende på typen av nät och nätets hastighet kan timeout tiderna behöva korrigeras. Det görs med ResendTime attributen in NodeConfig och FriendNodeConfig objekten.

Se dokumentet QCOM för mer info.

Näthanterare

Näthanteraren skickar information om volymer och objekt mellan noder, det can t ex gälla vilka volymer som en node äger, vilka föräldrar, barn eller syskon ett objekt har, eller innehållet i object och attribut. Via näthanterare läggs även upp prenumationer, dvs data som cyklisk skickas från en nod till en annan, vanligtvis från process-noder till operatörs-stationer för bildvisning. Processerna för näthanteraren är rt_neth, rt_neth_acp och rt_tmon.

Larmhanterare

Larmhanteraren skannar alla övervaknings-objekt på en nod, och skickar larm och händelser till olika utenheter, t ex larm och händelselista i operatörsmiljön. Dessa i sin tur skicka

meddelanden med larm-kvittens tillbaka till larmhanteraren. Exakt vilka larm och händlser som skickas till en utenhet beror på en urvalslista, som för operatörsmiljön anges i OpPlace objektet. Endast larm och händelser under angivna hierarkier skickas.
Processen för larmhantering, rt_emon, konfigureras med MessageHandler objektet.

Historisk datalagring

Den historiska datalagringen innebär att data skickas från process-noder och lagras i en databas som ofta ligger på en dedikerad server. På process-noden skannar rt_sevhistmon alla attribut som markeras för lagring, hämtar upp aktuella värden och skickar över dem till processen sev_server, som lagrar dem i en databas. När kurvor på historiska data ska visas, skickas en förfrågan för operatörsmiljön, och ett lämpligt urval av punkter hämtas upp ur databasen och skickas tillbaka till operatörsmiljön.
Se kapitel Datalagring

Web och app kommunikation

Web gränssnittet och Android applikationen hämtar information från realtidsdatabasen i ProviewR med hjälp av server-processerna rt_webmon, rt_webmonmh och rt_webmonelog. Den här kommunikationen konfigureras med WebHandler objektet.

Status server

Runtime monitorn och Supervision center hämtar information från status servern. Den här kommunikationen bygger på http och soap, och konfigureras med StatusSeverConfig objektet.

14.2 Remote

Remote-konceptet i ProviewR är ett sätt att standardisera kommunikationen med andra system. Det utgörs av ett antal transport program och ProviewR objekt som används för att implementera ett antal olika kommunikations-protokoll och hantera inkommande och utgående meddedanden. Remote är designat för att använda olika transport protokoll som TCP/IP eller BEA Message Queue, och olika hårdvara som t ex ethernet eller seriella linjer.
Det grundläggande syftet med Remote är att förse programmeraren med ett gränsnitt för kommunikation.

14.2.1 Introduktion

Ett antal olika klasser och objekt används för att hantera kommunikation.

RemoteConfig

Krävs för att någon remote kommunikation ska vara möjlig. Utan detta objekt startas inte remotehandlern. Ett RemoteConfig objekt placeras i nod-hierarkin.

RemNode

Definierar en länk till en remote nod av någon typ med ett specifikt protokoll.

Flera olika protokoll supportas and det finns en specifik klass för varje protokoll.
Det supportade protokollen är:

TCP/IP
UDP/IP
RabbitMQ
MQTT
MQ (BEA Message Queue)
ALCM (ett Digital protokoll, modell äldre, supportas av historiska skäl)
Serial
Modbus/RTU Serial
3964R (seriellt protokoll från Siemens)
Webspear MQ

Konfigurationen av varje protokoll beskrivs längre ner.
RemNode-objekt placeras under RemoteConfig-objektet.

RemTrans

Generell klass som denfinierar ett specifikt meddelande till en från en viss nod på ett visst protokoll. Ska placeras under ett RemNode-objekt. Storleken på det meddelande som ska skickas anges i RemTrans-objektet. Det data som ska sändas läggs i ett buffer-objekt som läggs som barn till RemTrans-objektet. När ett meddelande ska skickas hämtas data med den specifierade längden från buffer-objektet. I vissa fall adderas också en header till meddelandet.

Buffer

Definierar dataarean för sändning resp mottagning för ett meddelande. Buffer-objekt finns i olika storlekar, och ska placeras under RemTrans-objektet. Buffer-objektet måste var av minst den storlek som det meddelandet som ska skickas eller tas emot har. Om storleken inte räcker till kommer meddelandet att huggas av.

RemTransSend

Funktions-objekt som används i plc-program för att skicka meddelanden.

RemTransRcv

Funktions-objekt som används i plc-program för att ta emot meddelanden.

Loggning av transaktioner

Remote transaktioner kan loggas på en text-fil. Hur mycket som ska loggas konfigureras i RemTrans objektets LoggLevel attribut, och logfilen konfigureras med ett LoggConfig objekt.

14.2.2 Protokoll

Protokollet specifieras med den typ av RemNode-objekt som används vid konfigureringen. För varje RemNode objekt som är konfigurerat, startas en transport process, dvs ett program som hanterar det valda protokollet. Processen kommer att hantera alla de RemTrans objekt som ligger som barn till RemNode-objektet.

14.2.2.1 UDP

UDP använder socket kommunikation utan länk (datagram), till skillnad från TCP som ett länkat

protokoll. I RemNodeUDP-objektet anges ip-adressen till den nod man ska kommunicera med och port-nummer in båda ändarna. Det lokala port-numret måste vara unikt på noden.

Meddelanden sänds med en speciell header som adderas till data bufferten. Headern läggs i början på meddelandet. Headern innehåller information om det sända meddelandet, och används för att adressera meddelandet till rätt buffer-objekt. Headern ser ut så här:

```
char       RemId1;  /* STX (Hex 02) */
char       RemId2;  /* ETB (Hex 0F) in data message without acknowledge
                       ENQ (Hex 05) in data message with acknowledge
                       ACK (Hex 06) in acknowledge message */
short int  Length;  /* Number of bytes in message including this header */
short int  MessId1; /* Message identity part 1 */
short int  MessId2; /* Message identity part 2 */
```

Alla heltal i headern kommer skickas som big endian, dvs den mest signifikanta byten först. I användardelen av meddelandet är det konstruktörens ansvar att konverterar mellan big och little endian när detta behövs. Det går även att sända meddelanden utan header genom att sätta attributet DisableHeader till TRUE. Vid kommunikation mellan två ProviewR system ska headern finnas med. MessId1 och MessId2 hämtas från attributen RemTrans.Address[0] och RemTrans.Address[1]. Mha headern kan man även ange att kvittens ska skickas på meddelandet. Om kvittensen uteblir, upprepas sändningen av meddelandet cykliskt med det tid som ligger i RemTrans-objekts RetransmitTime attribut.

Eftersom UDP/IP är ett länk-fritt protokoll finns funktionen att övervaka kopplingen med keepalive-meddelanden. Detta anges med attributet UseKeepAlive.

Skicka meddelanden

Transporten kommer att skicka meddelanden till remote-porten, som består av header + data. MessId i headern hämtas från RemTrans.Address[0,1], byte-switchade och sända som big endian. Om MaxBuffer i remtrans-objektet > 0, sänds meddelandet med kvittens och lagras i återsändnings-kön för remnoden. När kvittensen tas emot, tas meddelandet bort ur återsändnings-kön. Detta utförs automatiskt av transport processen.

Ta emot meddelanden

När ett meddelande tas emot, kontrolleras först headern för att se att det är ett korrekt RemTrans meddelande. Sedan letar man upp det RemTrans-objekt med Address[0,1] som matchar det byte-switchade MessId-värdet. Om buffer-objektet är tillräckligt stort för meddelandet, lagras meddelandet där, och DataValid flaggan sätts. Om RemNode objekete har markerats för att användas utan header (DisableHeader-attributet är satt), letas efter en RemTrans, angiven som mottagare, med tillräckligt stort buffer-objekt.

14.2.2.2 TCP

RemnodeTCP konfigureras i stort sett som RemnodeUDP. Den stora (enda) skillnaden är att TCP är ett länk protokoll som agerar enligt klient/server modell. Alltså måste man antingen koppla upp mot en remote socket som klient, eller vänta på en uppkoppling som server. När man agerar som server, kommer enbart en klient att accepteras. ConnectionMode attributet i RemNode-objektet avgör om man är klient eller server. 0 (default värdet) anger klient och 1 anger server.

14.2.2.3 RabbitMQ

RemnodeRabbitMQ konfigurerar transporter med meddelandehanteraren RabbitMQ. Exchange, SendQueue och ReceiveQueue konfigureras i remnode-objektet. Köerna som inte exister

skapas av servern. Om Exchange inte är angiven används default exchange.

Username och password anges för att få tillgång till RabbitMQ servern. Användaren måste vara definerad i servern med lämplig åtkomst. Detta kan göras med rabbitmqctl på servern, t ex

```
> rabbitmqctl add_user myuser mypasswd
> rabbitmqctl set_permissions -p / myuser .* .* .*
```

I RemTrans-objektet används Address[0] och Address[1] för att addressera ett sänt meddelande till en viss RemTrans på mottagarnoden. Address[3] anger meddelandets 'delivery mode' där 2 är persistent och andra värden ej persistent. Address[2] är en option bitmask där bit 1 är KeepAll och bit 2 MsgOrder.

KeepAll betyder att meddelanden läggs tillbaka på kön om remtransen är upptagen. Normalt ignoreras de. Om det finns flera remtransar på samma kö, kan ordningen kastas om när meddelanden läggs tillbaka. Om MsgOrder biten är satt bibehålls ordningen. MsgOrder sätter prefetch count till 1.

Om ett meddelande ska återskapas efter ett server haveri, ska Durable vara satt i RemoteRabbitMQ-objektet, och 'delivery mode' i RemTrans.Address[3] ska sättas till 2 (persistent).

Om meddelanden ska kunna överleva nätverksavbrott är en lämplig metod att skicka meddelanden till en lokal rabbitmq-server och ta emot meddelanden från en server på sändande node. I det här fallet behövs olika RemnodeRabbitMQ-objekt för sändning och mottagning, och rabbitmq severn måste startas i båda noderna.

14.2.2.4 MQTT

RemnodeRabbitMQ konfigurerar meddelanden med MQTT.
Subscript och Publish konfigureras i remnode-objektet.

Username och password anges för att få tillgång till MQTT servern. Användaren måste vara definerad i servern med lämplig åtkomst. För Mosquitto kan en password fil genereras med mosquitto_passwd. Filen ska sedan kopieras till /etc/mosquitto.

I RemTrans-objektet används Address[0] och Address[1] för att addressera ett sänt meddelande till en viss RemTrans på mottagarnoden.

Specificationen av Topic skiljer sig åt om remoteheader är aktiverad eller inte.

Om header är aktiverad

- Sändning: publisering görs med topic i PublishTopic i RemnodeMQTT.
  Address[0] och Address[1] i remtrans-objektet används för att matcha remtrans-objekt.

- Mottagning: Prenumerationer görs med topic i SubscribeTopic i RemnodeMQTT-objektet.
  Meddelandet levereras till RemTrans-objektet med matchande Address[0] och Address[1].

Om header är inaktiverad

- Sending: publisering görs med topic i RemTrans.TransName.

- Mottagning: En generisk topic anges i SubscribeTopic i RemnodeMQTT-objektet, t ex 'lab57/rcv/#'.
  En mer avgränsande topic anges i RemTrans.TransName, t ex 'lab57/rcv/msg1'.

14.2.2.5 MQ

RemnodeMQ är en transport för att skicka meddelanden med BEA Message Queue (BMQ). Det krävs att BEA Message Queue är installerad på noden. BMQ är ett säkert sätt att skicka meddelanden till andra noder även om noden momentant inte är uppe, och även för att säkerställa att meddelanden

levereras säkert till noden själv.

Det här avsnittet kräver grundläggande kunskap om BEA Message Queue. Kortfattat kan man säga att kommunikationen går på en speciell bus, där varje nod har ett grupp-nummer och kan enbart kommunicera med andra grupper på samma bus. På varje grupp kan flera köer konfigureras.

För att kunna starta transporten måste naturligvis Message Queue vara startad.
Man måste även sätta några omgivningsvariabler. Dessa är:

```
DMQ_BUS_ID
DMQ_GROUP_ID
DMQ_GROUPNAME
```

I RemnodeMQ-objektet konfigurerar man vilken BMQ-kö som tar emot meddelanden i attributet MyQueue. Man konfigurerar även remote-nodens grupp-nummer och kö i attributen TargetGroup och TargetQueue.

Skicka meddelanden

I likhet med UDP och TCP-transporterna används RemTrans.Address[0,1] till att identifiera meddelanden. Address[0] representerar meddelande klass och Address[1] meddelandetyp (enligt BMQ nomenklatur). Address[2,3] används för att definiera vilken typ av leverans-mod (Address[2]) som ska användas och vad som ska göras om meddelandet inte kan levereras (Address[3]).

Möjliga leverans-moder är:

```
PDEL_MODE_WF_SAF 25
PDEL_MODE_WF_DQF 26
PDEL_MODE_WF_NET 27
PDEL_MODE_WF_RCM 28
PDEL_MODE_WF_MEM 29
PDEL_MODE_AK_SAF 30
PDEL_MODE_AK_DQF 31
PDEL_MODE_AK_NET 32
PDEL_MODE_AK_RCM 33
PDEL_MODE_AK_MEM 34
PDEL_MODE_NN_SAF 35
PDEL_MODE_NN_DQF 36
PDEL_MODE_NN_NET 37
PDEL_MODE_NN_RCM 38
PDEL_MODE_NN_MEM 39
PDEL_MODE_WF_DEQ 40
PDEL_MODE_AK_DEQ 41
PDEL_MODE_WF_CONF 42
PDEL_MODE_AK_CONF 43
PDEL_MODE_WF_ACK 44
PDEL_MODE_AK_ACK 45
```

och möjliga alternativ om meddelandet inte kan levereras:

PDEL_UMA_RTS 1
PDEL_UMA_DLJ 2
PDEL_UMA_DLQ 3
PDEL_UMA_SAF 4
PDEL_UMA_DISC 5
PDEL_UMA_DISCL 6

Alla kombinationer är inte tillåtna (se BEA Message Queue dokumentationen för mer info).

Följande kombinationer rekommenderas,

för säker leverans av meddelanden
Address[2] = 26
Address[3] = 4

och för att kasta meddelanden som inte kan levereras
Address[2] = 39
Address[3] = 5

Om Address[2,3] båda har satts till 0 kommer en default uppsättning att användas, som kastar meddelanden som inte kan levereras.

Ta emot meddelanden

Address[0,1] används för att idenfiera meddelandet. Address[0] representerar meddelande-klassen och Address[1] meddelande-typen (enligt BMQ-nomenkatur).

14.2.2.6 Seriell

RemoteSerial är ett försök för generall användning av enkla seriella kommunikations-protokoll. Det är användbart när man har en envägs sändning av meddelanden från någon typ av utrustning till styrsystemet. Man kan specifiera upp till åtta terminierings tecken i attributet TermChar[0-7]. Terminerings-tecknen används för att detektera slutet på ett mottaget meddelande (dvs ett tecken matchar något av terminieringstecken). Specificera också inställningen för den seriella porten, dvs DevName, Speed, Parity (0 = ingen, 1 = udda, 2 = jämn), StopBits och DataBits. De kan t ex vara /dev/ttyS0, 9600, 0, 1, 8.

14.2.2.7 3964-R

3964R är ett enkelt seriellt kommunikations-protokoll utvecklat av Siemens.

Man specificerar inställningen för den serialla porten med RemnodeSerial, med undantag för stop bitar. Man måste även ange tecken timeout (maximal tid mellan ankomna tecken) i attributet CharTimeout. I AckTimeout attributet anges tiden man väntar för ett svar.

Meddelandne skickas rak på utan header. ACK, NAK, DLE och BCC hanteras enligt 3965R protokollet.

Ta emot meddelanden

Det kan endast finnas ett RemTrans objekt för mottagning pga att det saknas header i protokollet. Varje mottaget meddelande läggs i första bästa RemTrans objekt under RemNode objektet. Om buffer objektet är tillräckligt stort för meddelandet lagras meddelandet i buffer-objektet och DataValid flaggan sätts.

Skicka meddelanden

Transporten sänder 3964R meddelanden utan att addera någon header.
Om man inte har kontakt med den andra noden kommer meddelandet att buffras, förutsatt att det finns fria buffrar för meddelandet.

14.2.2.8 Modbus Serial

MODBUS formatet som är implementerat är RTU. För att identifiera meddelanden används fälten för slav-adress och funktions-kod i MODBUS headern. I RemTrans[0] och [1] anges dessa värden i ProviewR världen. Se MODBUS specifikationen för mer info.

Modbus Serial är ännu inte implementerat som ett I/O system i ProviewR. Man måste konfigurera

meddelandena genom att använda RemnodeModbus och specifiera RemTrans-objekt för de olika operationerna man vill utföra. Modbus arbetar i en fråga/svara mod så att för varje operation som ska utföras, specifieras ett RemTrans-objekt för sändning och ett för mottagning. Modbus-headern i meddelandet och checksummehanteringen görs automatisk, men man måste specifiera innehållet i meddelandet för inläggning till sänd-bufferten. På samma sätt måste man avkoda innehållet i mottagna meddelanden.

Modbus TCP är implementerat som ett I/O system i ProviewR, Se mer information om detta i dokumentet "Guide to I/O Systems". Med Modbus TCP behöver man inte hantera avkodning av meddeladen.

Ta emot meddelanden

När ett meddelande tas emot, letas efter den RemTrans med Address[0] och [1] som matchar fälten för slav-adress och funktions-kod i meddelandets header. Om buffer-objektet är tillräckligt stort, lagra meddelandet och DataValid flaggan sätts.

Skicka meddelanden

Meddelanden som sänds använder innehållet i RemTrans.Address[0] och [1} för fälten för slav-adress och funktions-kod i meddelandets header.

14.2.2.9 Websphear MQ

RemnodeWMQ är en transport för att sända meddelanden med Websphear Message Queue (WMQ). Det kräver att Websphear MQ är installerat på noden.

RemnodeWMQ konfigurerar kommunikation genom en meddelande-kö och använder Websphear MQ som klient. Allt som har med Server, kanal, kö-manager och kö att koppla till konfigureras i RemnodeWMQ-objektet.

Konfigurationen för de olika meddelandena konfigureras i RemTrans objekt som definierar varje in och utgående meddelande. För varje RemTrans-meddelande konfigureras följande:

TransName	en sträng som definierar MsgId (identitet för meddelandet).
Address[0]	definierar om meddelandet ska sändas som ett 'persistent' meddelande eller inte, dvs loggas på disk eller inte. 1 betyder att det kommer att sändas som 'persistent'.

14.2.3 Ett exempel

För att visa hur man arbetar med de klasser som översikligt beskrivits ovan, startar vi med ett litet exempel. Klasserna beskrivs med detaljerat nedan.

I vårt exempel har vi ett ProviewR system som kommunicerar med en annan nod via UDP/IP. Vi vill skicka ett antal meddelanden i båda riktningarna. Min nod heter 'dumle' och remote noden heter 'asterix'.

Meddelanden vi ska sända är:

d_a_RequestData	4 Byte
d_a_Report	20 Byte

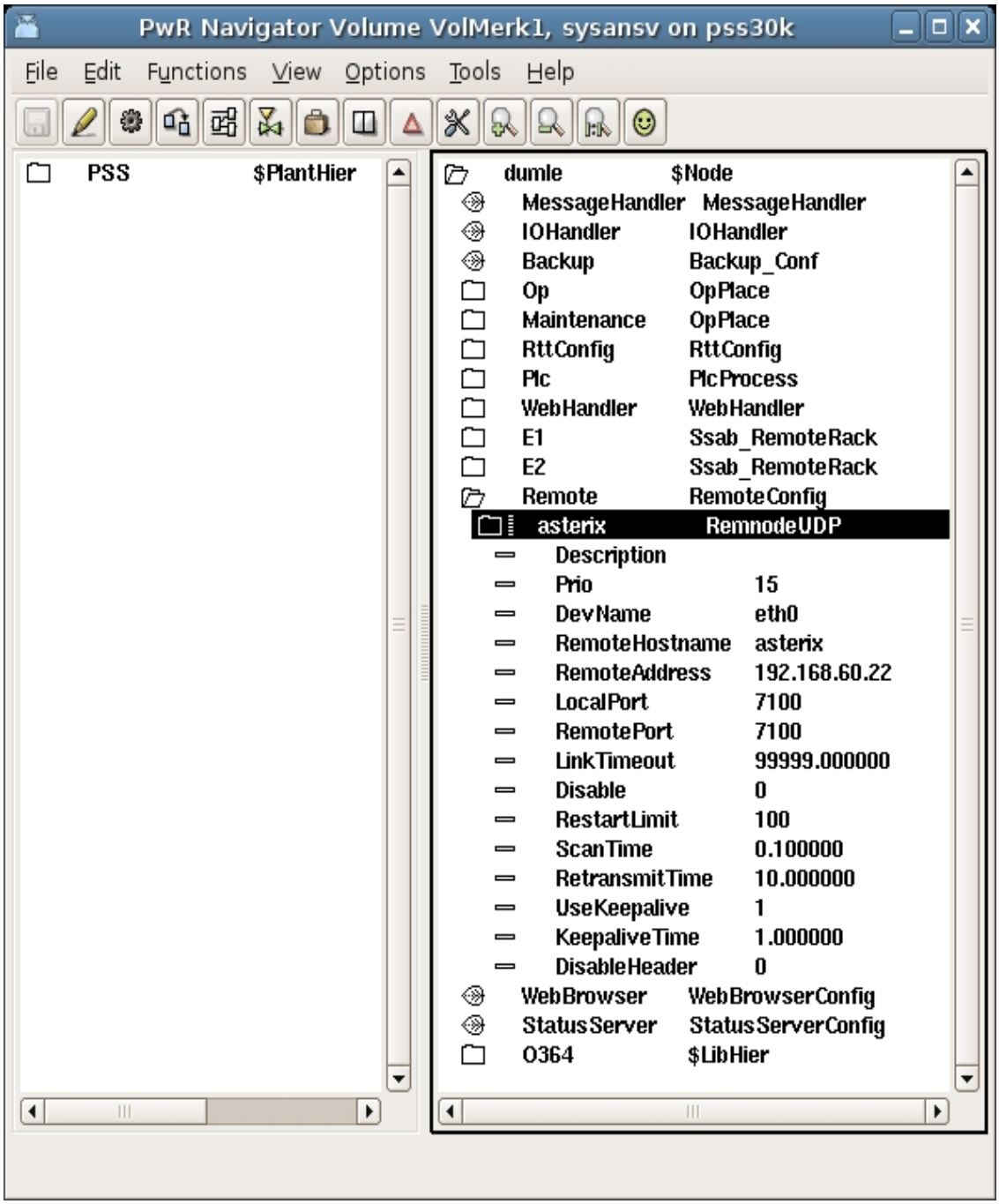
och meddelanden som vi ska ta emot:

a_d_Data	365 Byte (as an answer to the d_a_RequestData-message)
a_d_Error	10 Byte

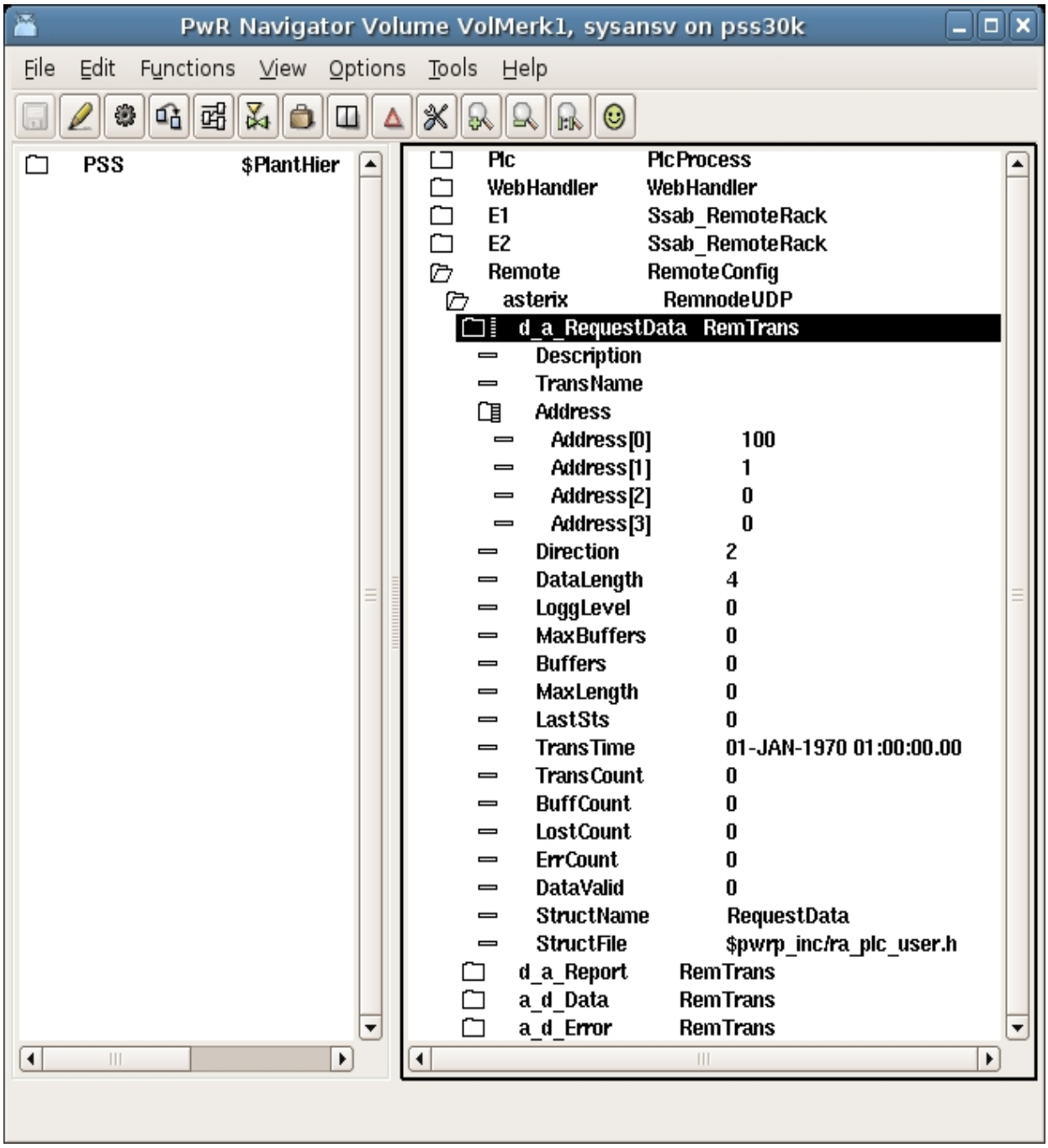
Konfigurationen i nod-hierakin ser ut så här.

Under RemoteConfig-objektet ligger ett RemnodeUPD objekt, där IP-address, nodnamn och

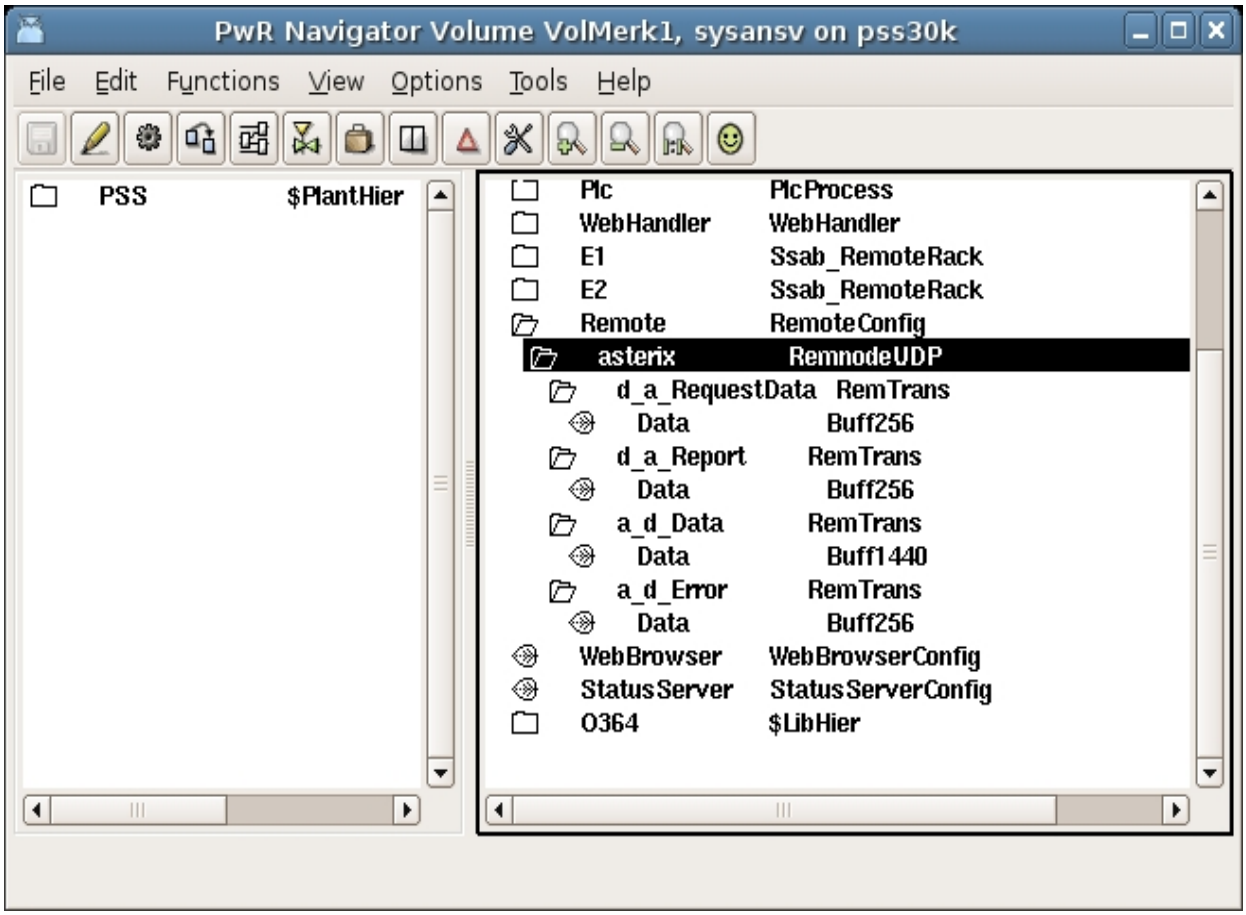
port-nummer är konfigurerade.



Under RemnodeUPD-objeketet ligger fyra RemTrans objekt, ett för varje meddelande. I remtrans-objekten har riktningen (send eller receive) konfigurerats. Adresserna är numrerade för att kunna särskilja meddelandena, och meddelandenas storlek är angiven.



Under remtrans-objektet ligger buffer-objektet. För de mindre meddelandena används Buff256 objekt. Data meddelandet är större och använder ett Buff1440 objekt.



Data-strukturen

Data-strukturen för meddelandena är definierad i filen ra_plc_user.n i $pwrp_inc katalogen. Den här filen inkluderas automatiskt vid kompilering av plc-koden. Data-strukturen ser ut så här:

```
typedef struct {
  pwr_tUInt32  Id;
} d_a_RequestData;

typedef struct {
  pwr_tUInt32   Id;
  pwr_tFloat32  data_1;
  pwr_tInt32    data_2;
  pwr_tInt32    data_3;
  pwr_tInt32    data_4;
} d_a_Report;

typedef struct {
  pwr_tUInt32   Id;
  pwr_tFloat32  data_1;
  ...
  ...
  pwr_tInt32    data_xx;
} a_d_Data;

typedef struct {
```

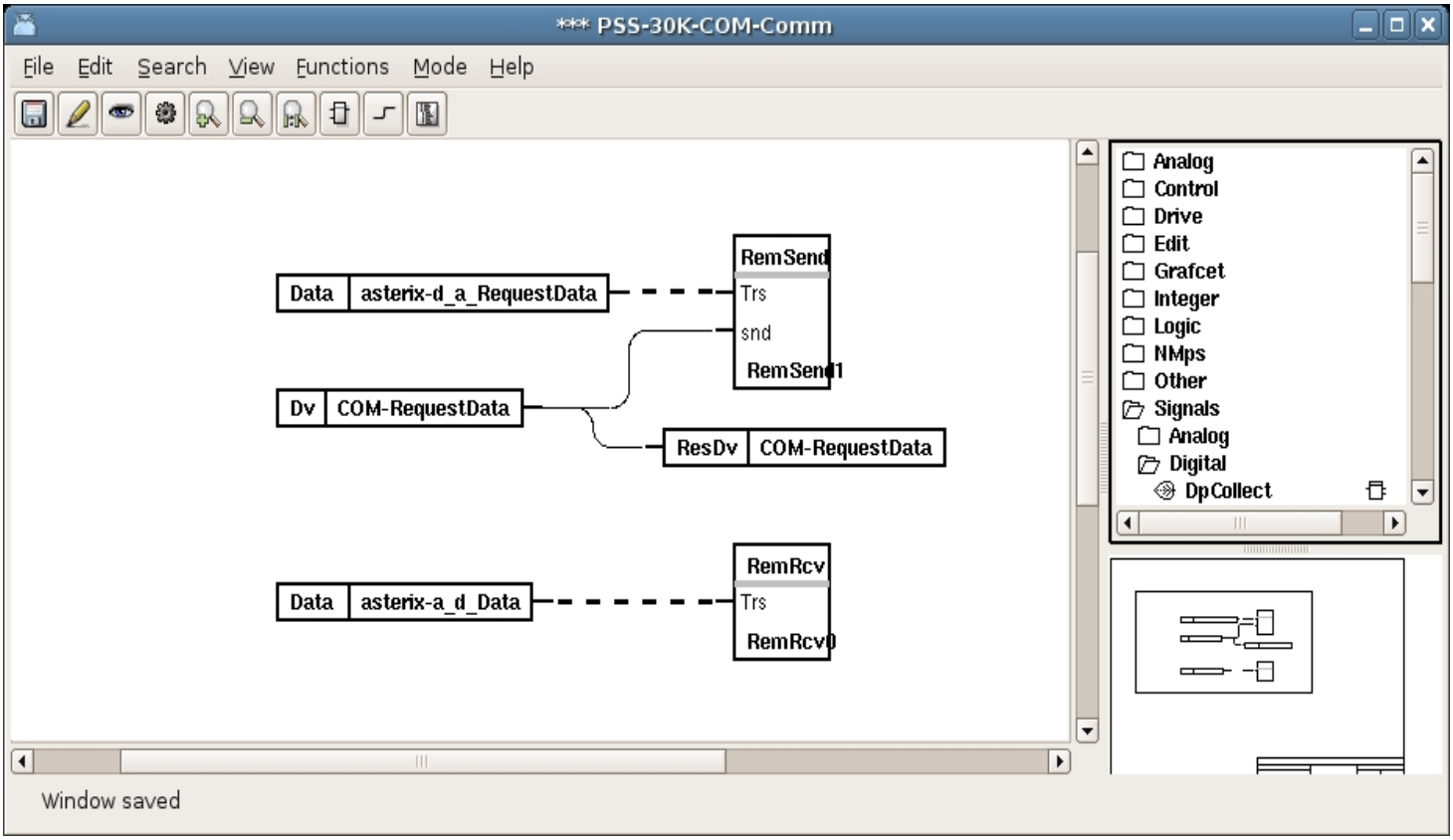
```
  pwr_tUInt32   Id;
  pwr_tInt32    func_no;
  pwr_tInt16    err_code;
} a_d_Data;
```

För att optimera läs och skrivning kommer gcc kompilatorn normalt att placera element i data-strukturer på 32 eller 64-bit ordgräns. När man skapar strukturer för kommunikation kommer det att skapa oreda då olika kompilatorer och plattformar har olika regler för alignment. För att undvika detta, använd 'attribute packed' vid deklarationen av data-strukturen.

```
typedef struct {
  pwr_tUInt32   Id;
  pwr_tInt32    func_no;
  pwr_tInt16    err_code;
} __attribute__((__packed__)) a_d_Data;
```

Plc-koden

Koden ligger i ett PlcPgm med namnet Comm. I programmet finns ett RemTransSend-objekt och ett RemTransRcv objekt. Dess objekt återfinns under mappen 'Other' i paletten i plc-editorn. Till RemTransSend-objektet kopplas den RemTrans som ska sändas. I det här fallet d_a_RequestData meddelandet. Meddelandet sänds när Dv-signalen "RequestData" sätts. På samma sätt är a_d_Data meddelandet kopplat till RemTransRcv-objektet som tar emot svarsmeddelandet på request meddelandet.

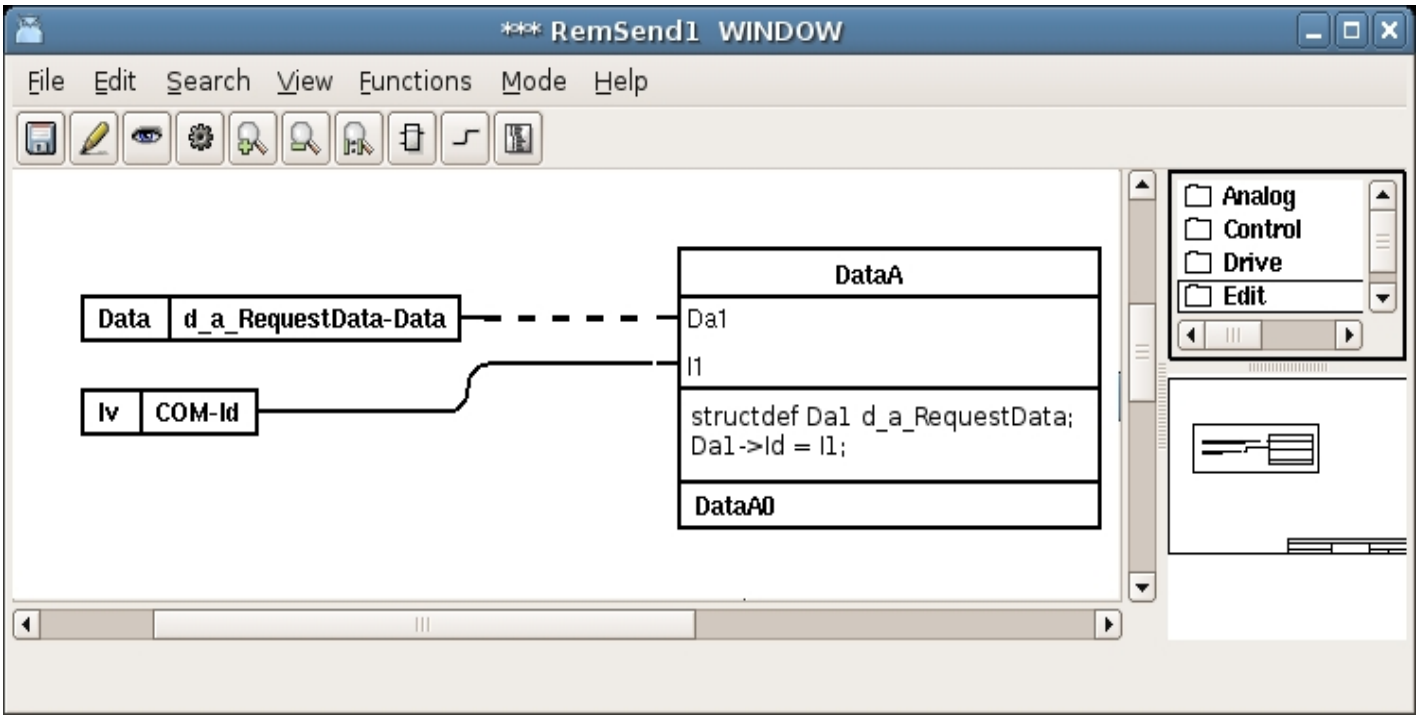


Både RemTransSend och RemTransRcv-objekten har underfönster. För RemTransSend, kommer underfönstret att exekveras vid en positiv flank på snd-ingången. När underfönstret har exekverats, sätts DataValid attributet i det kopplade RemTrans-objektet, och transport-processen för den aktuella Remnoden skickar iväg meddelandet och sätter DataValid till 0.

För RemTransRcv exekveras underfönstret när DataValid attributet i det kopplade RemTrans-objektet sätts. När transport-processen för den aktuella Remnoden tar emot ett meddelande, fyller den i buffer-objektet med meddelandet och sätter DataValid flaggan. När underfönstret har exekverat återställs DataValid flaggan.

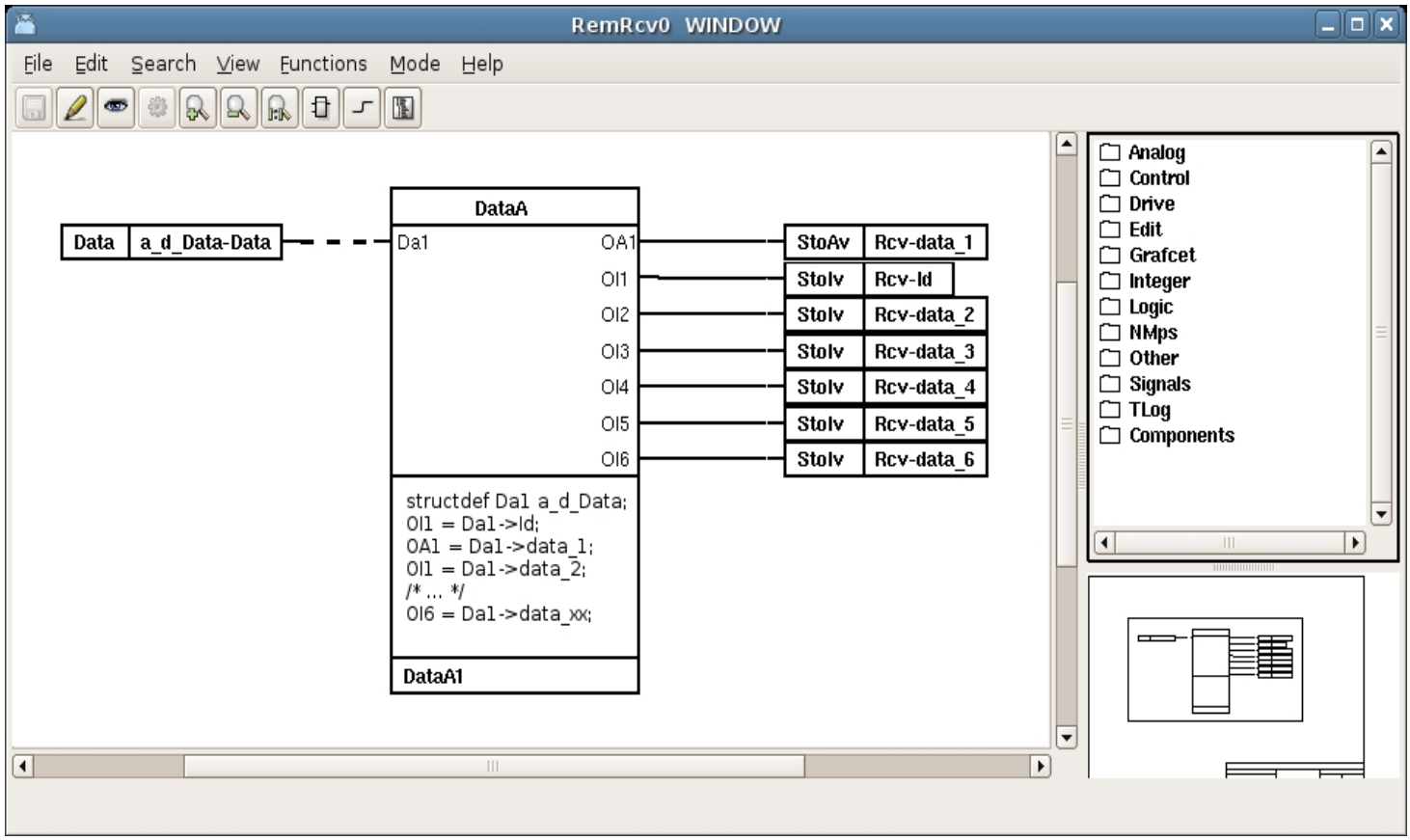
Underfönstret för sändning

I send underfönstret fyller man i data för sänd-bufferten. Buffer-objektet för meddelandet kopplas till en DataArithm. Den speciella 'structdef'-syntaxen castar Da1 till en pekare till en d_a_RequestData struct.



Underfönstret för mottagning

I mottagnings-underfönstret packas det mottagna meddelandet upp. Buffer-objektet för mottagningen kopplas till en DataArithm. Återigen använder vi classdef-satsen för att casta Da1 pekaren.
Data packas upp och läggs på DataArithm-objektets utgångar. Om inte utgångarna för en DataArithm räcker till adderar man flera DataArithm-objekt och fortsätter på samma sätt.


RemRcv0 WINDOW
File Edit Search View Functions Mode Help
Data
a_d_Data-Data
DataA
Da1
OA1
OI1
OI2
OI3
OI4
OI5
OI6
StoAv
Rcv-data_1
StoIv
Rcv-Id
StoIv
Rcv-data_2
StoIv
Rcv-data_3
StoIv
Rcv-data_4
StoIv
Rcv-data_5
StoIv
Rcv-data_6
structdef Da1 a_d_Data;
OI1 = Da1->Id;
OA1 = Da1->data_1;
OI1 = Da1->data_2;
/* ... */
OI6 = Da1->data_xx;
DataA1
Analog
Control
Drive
Edit
Grafcet
Integer
Logic
NMps
Other
Signals
TLog
Components

15 Datalagring

Det finns tre typer av datalagring i ProviewR, trendkurvor, fastkurvor och historisk lagring. Trendkurvurna lagrar cycliskt data under en kortare period i realtidsdatabasen.
Fastkurvor triggas av någon händelse och lagrar data under en period efter triggningen, och i vissa fall även före. Historiska lagringen sker på disk i en relationsdatabas och kan cykliskt lagra data under flera år.

15.1 Trendkurvor

Trendkurvor finns i två varianter
- DsTrend med en intern databuffer som kan lagra ca 500 mätvärden, och med en lägsta scantid på 1 s.
- DsTrendCurve, med konfigurerbar bufferstorlek och med en lägsta scantid på 20 ms.

15.1.1 DsTrend

DsTrend objektet har en intern databuffer på 1912 byte som t ex kan lagra 478 mätvärden av typen Float32. Även andra datatyper är möjliga. Konfigureringen är lite udda eftersom bufferten är uppdelad i två delar, men normal behöver man endast ange det attribut som ska lagras i DataName, och eventuellt ett värde i Multiple för att ange scantiden.

Se även DsTrend i Object Reference Manual.

15.1.2 DsTrendCurve

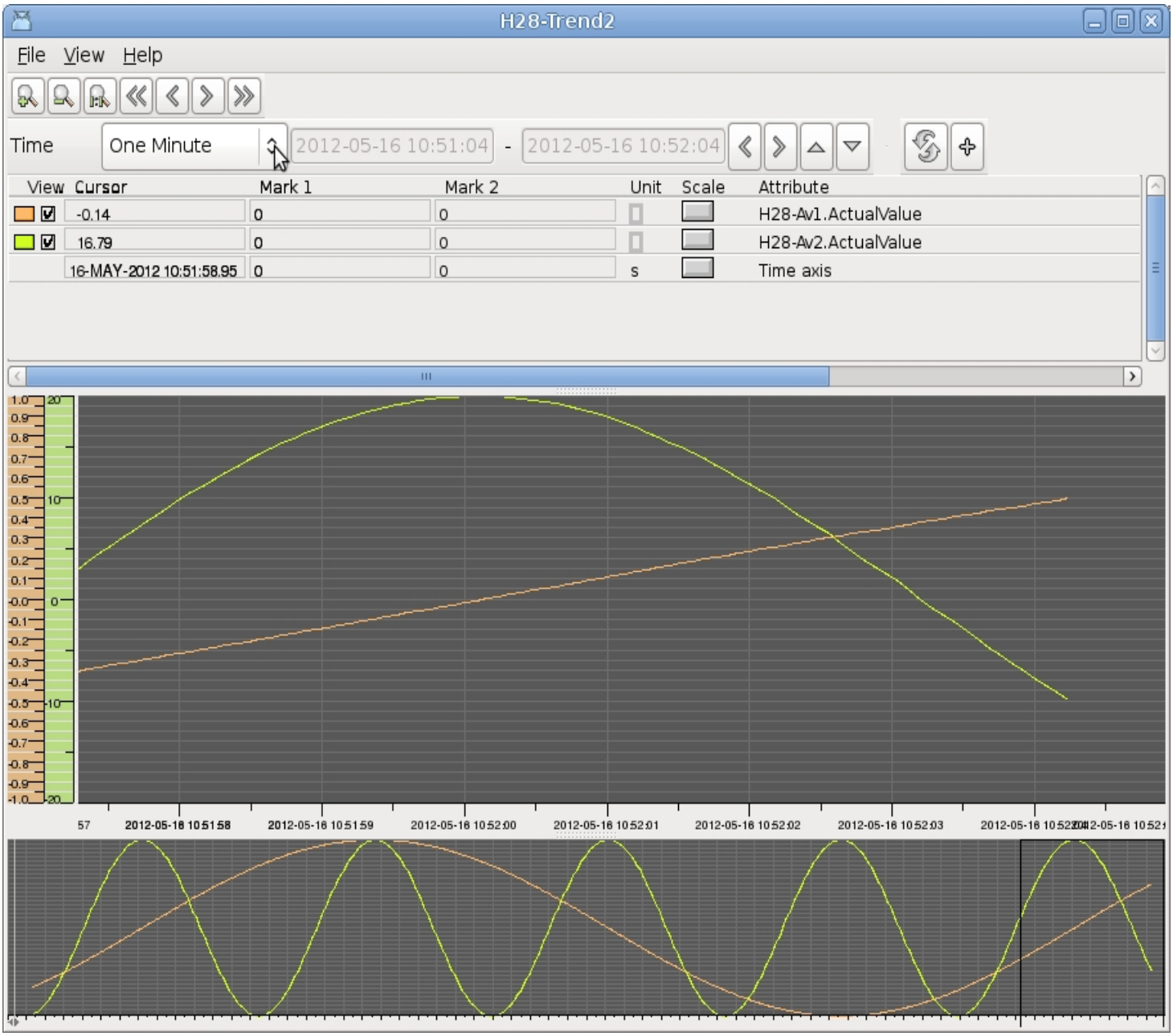


Fig Trendkurva med DsTrendCurve

DsTrendCurve objektet lagrar trendkurvor med externa bufferobjekt, där storleken på bufferobjekten begränsar antalet mätvärden som kan lagras. Ett CircBuff100k som i exemplet nedan kan innehålla 25000 mätvärden. Lagringen kan ske med scantider ner till 20 ms. Det finns också en snapshot funktion där den aktuella kurvan lagras och visas i ett fönster där den kan analyseras i detalj. Snapshot kurvan kan lagras på fil och öppnas vid ett senare tillfälle.

Trendkurvan konfigureras med ett DsTrendCurve objekt. 10 olika attribut som ska lagras kan anges i Attribute vektorn. För varje attribut ska en lagrings-buffer av typ CircBuffer skapas och anges i Buffer attributet. Storleken på bufferobjektet ska anpassas till antalet mätvärden. Också en tidbuffer kan anges, men denna behövs endast om snapshot funktionen ska användas. Tidupplösningen bestämmer storleken på tidsbufferten. Med en upplösning på 1 sekund, används 4 bytes per mätvärde, och men en upplösning på nanosekund 8 bytes.

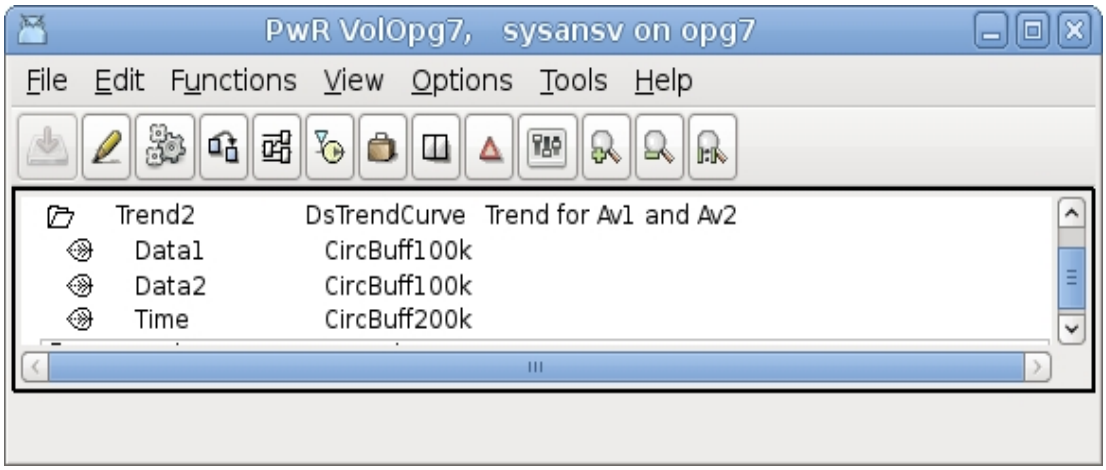


Fig Konfigurering av DsTrendCurve med bufferobjekt

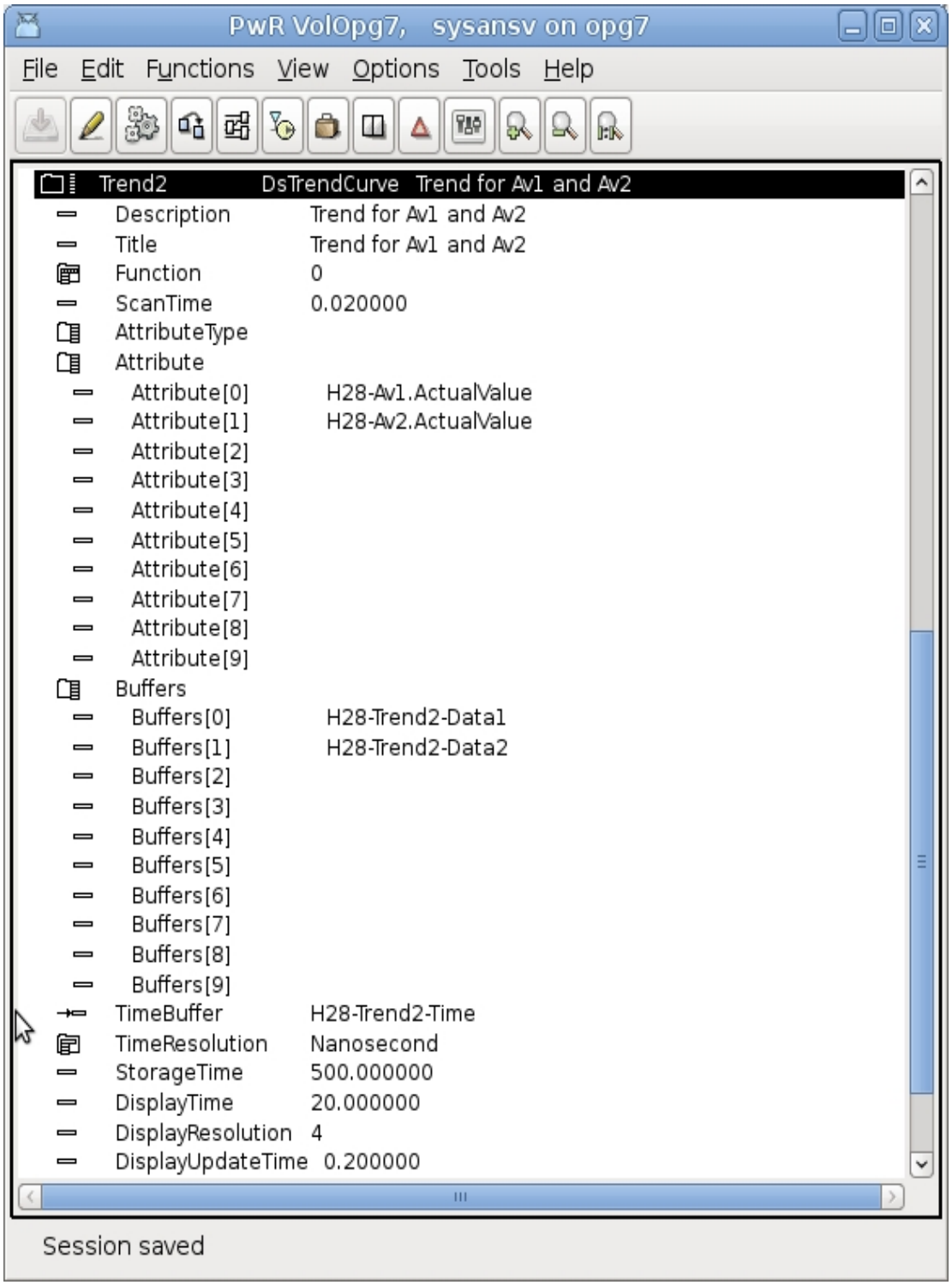


Fig Konfigurering av DsTrendCurve

Snapshot innebär att man tar en kopia av den lagrade datamängden och visar upp i ett separat fönster. Man kan konfigurera så att trendkurvan bara visar en begränsad del av den lagrade datamängden och att det finns data lagrade för en längre tid, och även med högre upplösning. I Snapshot fönstret kan man då gå längre tillbaka i tiden eller zooma in och öka upplösningen på kurvan.

Attributen DisplayTime och DisplayResolution i DsTrendCurve objektet används för att specificera den del av datamängden som ska visas i trend-fönstret. I exemplet ovan är det totala lagringstiden (StorageTime) 500 s, medan det endast är de 20 sista sekunderna

(DisplayTime) som visas i trendfönstret. DisplayResolution är satt till 4, vilket innebär att var fjärde mätvärde visas. Eftersom scantiden är 20 ms visas ett värde varje 80 ms, och total visas 20 s/0.08s = 250 värden. Hela datamängden innehåller 500 s/0.02 s = 25000 värden vilka alltså blir tillgängliga i snapshot fönstret.

Se även DsTrendCurve i Object Reference Manual.

15.2 Fast kurvor

En Fast kurva triggas av en viss händelse och lagrar sedan data under en viss tid som sedan visas upp i ett kurvfönster. Trigg-villkoret kan vara en digital signal som blir hög, ett analog värde som passerar en nivå, eller manuell triggning. Fastkurvan kan konfigureras att kontinuerligt lagra data så att även data före triggpunkten kan visas upp.

En fast kurva konfigureras med ett DsFastCurve objekt. Upp till 10 attribut kan anges i Attribute vektorn, och för varje attribut ska ett buffer-objekt skapas och anges i Buffers attributet. Även ett buffer-objekt for tiden ska skapas och anges i TimeBuffer attributet. Storleken på på bufferobjekten ska anpassas till den datamängd som ska lagras.

En fast-kurva kan visas i det ordinare kurvfönstret, som t ex öppnas med Fast alternativet i popup-menyn. Det kan även visas i en Ge graph med en FastCurve komponent.

Se även DsFastCurve i Object Reference Manual.

15.3 Historisk datalagring

Historisk lagring av processdata innebär att signaler och andra data cykliskt lagras i en databas, och därifrån kan hämtas upp för att visas i kurvor eller analyseras i andra sammanhang, t ex för prediktivt underhåll eller processutveckling.

Läs mer i Handbok i Processhistorik.

16 Applikationsprogrammering

Det här kapitlet handlar om hur man skriver applikations program, dvs program i c, c++ eller java som knyter upp sig mot ProviewR. Det förutsätts all läsaren har grundkunskaper i dessa språk.

För många tillämpningar går det utmärkt att koda all styrning grafisk med hjälp av plceditorn. Men det finns även tillämpningar som med grafisk programmering blir onödigt komplexa som t ex avancerade modeller, databashantering och materialstyning. Man skriver då ett applikationsprogram i c, c++ eller java som knyter upp sig mot realtids databasen rtdb. Programmet läser indata från rtdb, gör sina beräkningar och sätter utdata i rtdb där det t ex vidareprocessas av plc-programmet, skickas ut till I/O systemet och visas upp i operatörsbilder.

c++ är det programmeringsspråk som är vanligast vid applikations programmering och även mest utbyggt, vi kommer därför att koncentrera oss på detta. De funktioner som används finns närmare beskrivna i Programmer's Reference Manual (PRM).

16.1 Knyta upp sig mot databasen och hantera objekt och data

Vi börjar med att skriva ett enkelt c++ program som knyter upp sig mot realtidsdatabasen och länkar sig mot några objekt.

cpp-filen bör ligga på $pwrp_src katalogen, eller på på någon underkatalog till denna. Vi skapar katalogen $pwrp_src/myappl och editerar filen ra_myappl.cpp.

Datatyper

I include-filen pwr.h finns de grundläggande datatyperna i ProviewR definierade. De vanligaste är pwr_tBoolean som används för digitala signaler och pwr_tFloat32 för analoga signaler, men där finns även c-typer för alla andra ProviewR-typer, t ex pwr_tInt32, pwr_tUInt32, pwr_tString80 etc.

Initiering av gdh

Uppknytningen mot databasen görs med anropet gdh_Init() där man skickar med en sträng som identifierar applikationen. Först inkluderar vi pwr.h som innehåller grundtyperna i ProviewR och rt_gdh.h som innehåller gränssnittet mot realtidsdatabasen.

```
#include "pwr.h"
#include "rt_gdh.h"

int main() {
  pwr_tStatus sts;

  sts = gdh_Init( "ra_myappl");
```

```
  if ( EVEN(sts)) {
    cout << "gdh_Init failure " << sts << endl;
    exit(0);
  }
}
```

Funktionen returnerar en status variabel av typen pwr_tStatus. En jämn status innebär att något är fel, udda att anropet gick bra. Status kan även översättas till en sträng som ger mer information om vad som är fel, vilket beskrivs närmare längre fram.

Läsa och skriva värden

Om vi vill skriva och läsa ett attribut i ett objekt kan man använda funktionerna gdh_SetObjectInfo() och gdh_GetObjectInfo().

En läsning resp skrivning av Dv'n H1-H2-Start kan se ut så här. Notera att värdet av Dv'n ligger i attributet ActualValue

```
pwr_tBoolean value;

sts = gdh_GetObjectInfo( "H1-H2-Start.ActualValue", &value, sizeof(value));
if (ODD(sts)) {
  value = !value;
  sts = gdh_SetObjectInfo( "H1-H2-Start.ActualValue", &value, sizeof(value));
}
```

Direktlänka mot attribut

Ofta är ligger applikationsprogram i en oändlig loop och övervakar attribut i databasen och reagerar på vissa förändringar. Då är det bättre att direktlänka sig mot attributet, dvs skaffa en pekare. Det gör man med gdh_RefObjectInfo(). Ofta delar man upp programmet i en init() funktion som direktlänkar till attribut, en scan() funktion som som innehåller övervakning och styrfunktioner, och en close() funktion tar ner direktlänkningarna.

```
class ra_myappl {
  pwr_tBoolean *start_ptr;
  pwr_tRefId dlid;
 public:
  ra_myappl() {}
  void init();
  void scan();
  void close();
 };

void ra_myappl::init()
{
  sts = gdh_RefObectInfo( "H1-H2-Start.ActualValue", &start_ptr, &dlid,
                          sizeof(*start_ptr));
  if ( EVEN(sts)) exit(0);
}

void ra_myappl::scan()
```

```
{
  for (;;) {
    if ( *start_ptr) {
      // Do something...
      cout << "Starting" << endl;

      *start_ptr = 0;

    }
    sleep(1);
  }
}

void ra_myappl::close()
{
  gdh_UnrefObjectInfo( &dlid);
}
```

I init() funktionen fylls pekaren start_ptr i så att den pekar på värdet för Dv'n H1-H2-Start i databasen.

Varning !

Notera att pekare i c kräver varsam hantering. Om man använder pekar-aritmetik eller indexerar i vektorer är det lätt hänt att man pekar fel och skriver på fel ställe i databasen. Detta kan ge upphov till fel som är svåra att hitta orsaken till.

Direktlänka mot objekt

gdh_RefObjectInfo() kan förutom att direktlänka mot enskilda attribut, också direklänka mot objekt och attributobjekt.

Antag att vi vill fylla i punkter i en kurva och visa upp i en bild. Vi vill direktlänka till objektet H1-H2-Curve som är av klassen XyCurve. Includefilen pwr_baseclasses.hpp innehåller en c++ klass, pwr_Class_XyCurve, för objektet.

```
#include <math.h>
#include "pwr.h"
#include "pwr_baseclasses.hpp"
#include "rt_gdh.h"

class ra_myappl {
  pwr_Class_XyCurve *curve_ptr;
  pwr_tRefId dlid;
 public:
  ra_myappl() {}
  void init();
  void scan();
  void close();
 };

void ra_myappl::init()
{
  pwr_tStatus sts;
```

```
  pwr_tOName name = "H1-H2-Curve";

  // Connect to database
  sts = gdh_Init( "ra_myappl");
  if ( EVEN(sts)) exit(0);

  // Direct link to curve object
  sts = gdh_RefObjectInfo( name, (void **)&curve_ptr, &dlid, sizeof(*curve_ptr));
  if ( EVEN(sts)) exit(0);
}

void ra_myappl::scan()
{
  for ( unsigned int i = 0;;i++) {
    if ( i % 5 == 0) {
      // Calculate x and y coordinates for a sine curve every fifth second
      for ( int j = 0; j < 100; j++) {
        curve_ptr->XValue[j] = j;
        curve_ptr->YValue[j] = 50 + 50 * sin( 2.0 *  M_PI * (j + i) / 100);
      }
      // Indicate new curve to graph
      curve_ptr->Update = 1;
    }
    else if ( i % 5 == 2)
      curve_ptr->Update = 0;
    sleep(1);
    if ( i > 360)
      i = 0;
  }
}

void ra_myappl::close()
{
  gdh_UnrefObjectInfo( dlid);
}


int main()
{
  ra_myappl myappl;

  myappl.init();
  myappl.scan();
  myappl.close();
}
```

Programmet kompileras och länkas med

```
> g++ -g -c  ra_myappl.cpp -o $pwrp_obj/ra_myappl.o -I$pwr_inc -DOS_LINUX=1
      -DOS=linux -DHW_X86=1 -DHW=x86
> g++ -g -o $pwrp_exe/ra_myappl $pwrp_obj/ra_myappl.o $pwr_obj/pwr_msg_rt.o
      -L$pwr_lib -lpwr_rt -lpwr_co -lpwr_msg_dummy -lrt
```

Senare ska vi se hur man använder make för att kompilera och länka.

Om vi öppnar objektsbilden för H1-H2-Curve objektet kan vi studera resultatet.

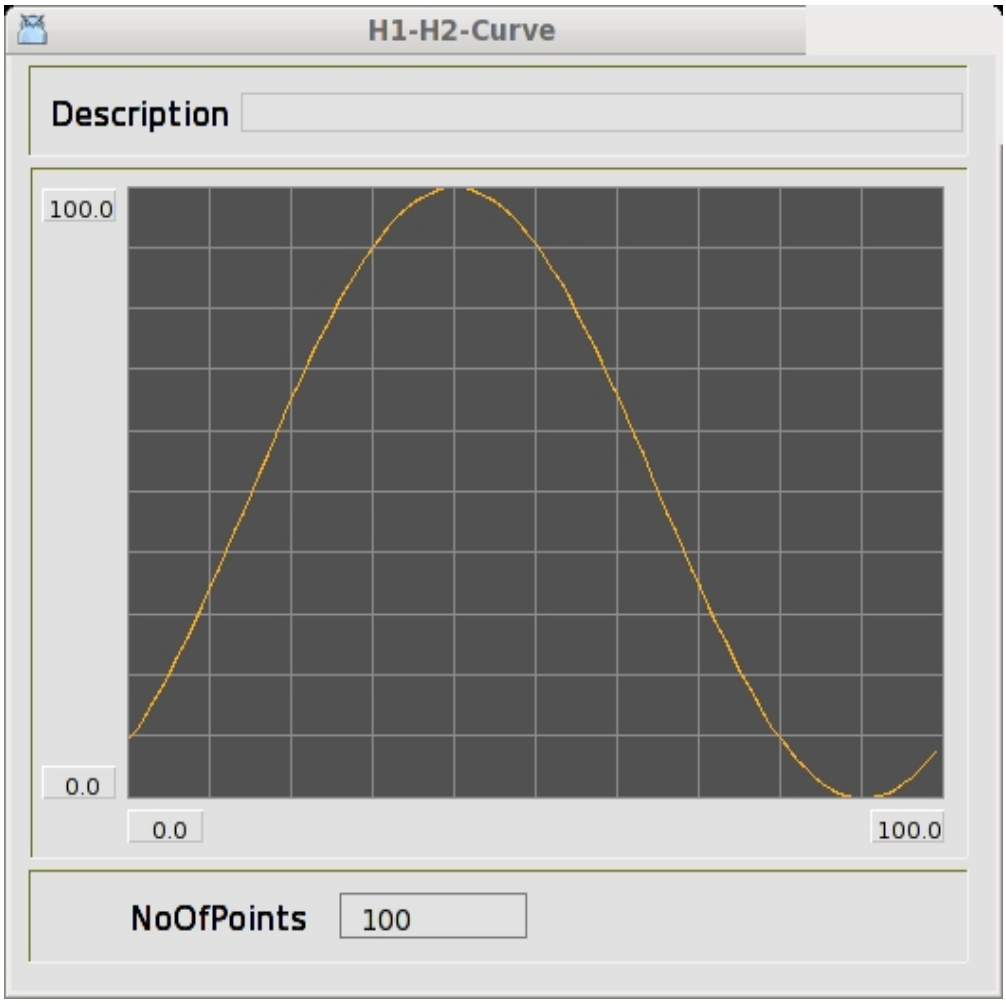


Fig Objektbild for kurvobjektet

16.2 Konsollogg

Logga på konsolloggen

Konsollog

Konsolloggen innehåller diverse loggning från systemprocesser. Om systemet går orent bör man titta på för att se om någon process loggar felmeddelanden. Den ligger som en textfil på $pwrp_log, pwr_'nodename'.log, men kan även öppnas från med rt_xtt från System/SystemMessages. Loggningarna har fem allvarlighetsnivåer fatalt, fel, varning, info och succé. Fatal och fel är rödfärgade, varning gulfärgad och info och succé grönfärgade.

Även applikationer kan skriva på konsollogen. Först initierar man konsollogen med errh_Init(), sedan kan man skriva meddelanden av olika allvarlighetsgrad med errh_Fatal(), errh_Error(), errh_Warning(), errh_Info() och errh_Success().

errh_Init() anropas före gdh_Init() och har som argument ett namn på applikationen samt ett applikationsindex som anges med errh_eAnix_Appl1, errh_eAnix_Appl2 etc. Varje applikation ska

ha ett unikt applikationsindex inom noden.

```
#include "rt_errh.h"

sts = errh_Init( "ra_myappl", errh_eAnix_Appl1);
```

Till logg funktionerna skickar man den sträng som ska skriva ut i loggen, t ex

errh_Error("Something went wrong");

Strängen kan även fungera som formatsats som kan innehålla %s för att formatera strängar, %f för flyttal och %d för heltal, se printf för mer info.

```
errh_Error( "Number is to high: %d", n);
```

Formatet %m översätter en statuskod till motsvarande text

```
catch ( co_error e) {
  errh_Error( "Error status: %m", e.sts());
}
```

Applikations status

Varje applikation har ett statusord i $Node objektet. Det ligger i attributet ProcStatus[] i elementet applikationsindex + 20. Detta ska återspegla applikationens tillstånd och sätts av applikationen själv med med funktionen errh_SetStatus().

```
errh_SetStatus( PWR__ARUN);
```

PWR__ARUN är definierad i rt_pwr_msg.h och länkad till texten "Application running".

Andra användbara status är

PWR__APPLSTARTUP "Application starting up" (info)
PWR__APPLRESTART "Application restaring" (info)
PWR__APPLTERM "Application terminated" (fatal)

I nodobjektet finns en systemstatus som är en slags summa av status för alla serverprocesser och applikationsprocesser. I systemstatus läggs den server- eller applikation-status som har den högsta allvarlighetsgraden.

Övervakning

En applikation som har anropat errh_Init övervakas av systemet. Den ska anropa funktionen aproc_TimeStamp() cykliskt, annars sätts applikationsstatus till "Process timeout" (fatal). Timeouttiden för applikaioner är 5 s.

Applikationobjekt

Man kan även skapa ett applikationsobjekt för applikationen. Detta läggs under i nodhierkin under $Node objektet och är av klassen Application.

Applikationsobjektet registreras med funktionen aproc_RegisterObject() som har objektsidentiteten för objektet som argument.

```
pwr_tObjid aoid;
pwr_tOName name = "Nodes-MyNode-ra_myappl";

sts = gdh_NameToObjid( name, &aoid);
if (EVEN(sts)) throw co_error(sts);

sts = aproc_RegisterObject( aoid);
if (EVEN(sts)) throw co_error(sts);
```

Statusbilden

Applikationen visas i statusbilden för noden, om applikationen har initierat errh och registrerat applikationsobjektet. Den kommer att visas under 'Application' på den rad som motsvara applikationsindex. I bilden visas status för applikationen och senaste/allvarligaste loggmeddelande.

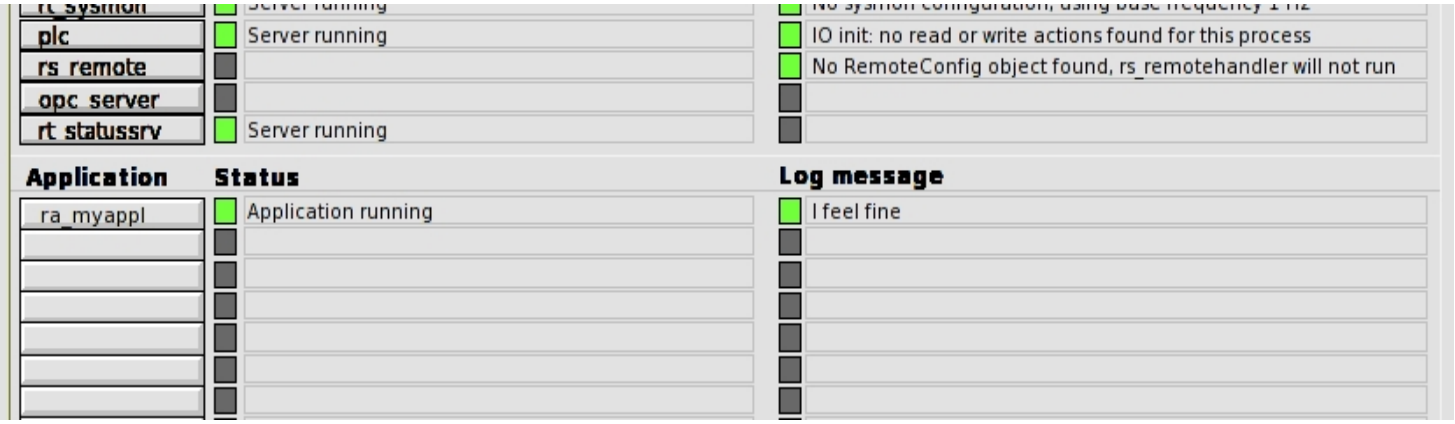


Fig Detalj ur nodbilden visar status och loggmeddelande för applikationen.

Om processen haltas sätts status till timeout, detta påverkar även systemstatus.

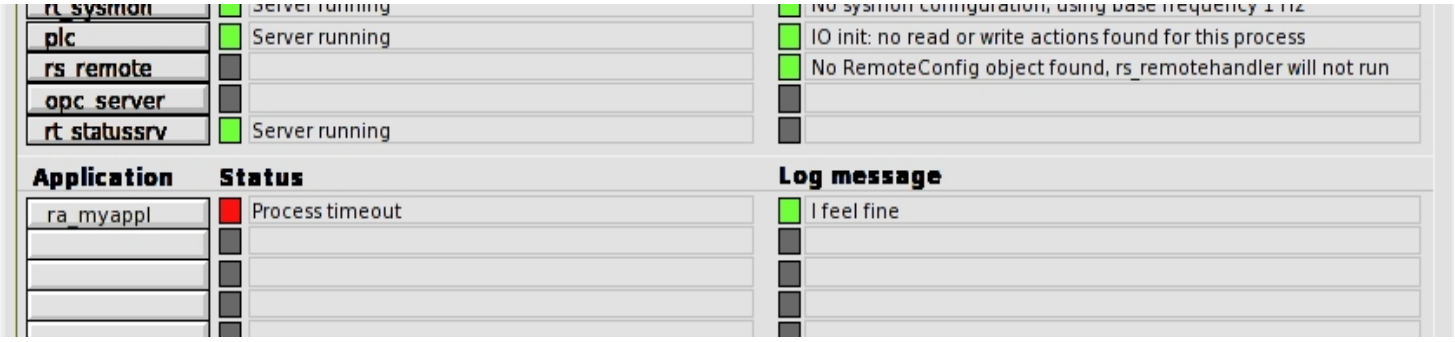


Fig Applikationen har haltats.

Exempel
I det här exemplet har vi jobbat vidare med programmet för xy-kurvan ovan, och fört in anrop för att sätta applikationsstatus, logga på konsolloggen och registrera applikationsobjektet.

```
#include <math.h>
#include <iostream>
```

```
#include "pwr.h"
#include "pwr_baseclasses.hpp"
#include "rt_gdh.h"
#include "rt_errh.h"
#include "rt_aproc.h"
#include "rt_pwr_msg.h"
#include "co_error.h"

class ra_myappl {
  pwr_Class_XyCurve *curve_ptr;
  pwr_tRefId dlid;
 public:
  ra_myappl() {}
  void init();
  void scan();
  void close();
};

void ra_myappl::init()
{
  pwr_tStatus sts;
  pwr_tOName name = "H1-H2-Curve";
  pwr_tObjid aoid;

  // Init errh with anix 1
  sts = errh_Init( "ra_myappl", errh_eAnix_appl1);
  if ( EVEN(sts)) throw co_error(sts);

  // Write message to consolelog and set application status
  errh_Info( "I feel fine");
  errh_SetStatus( PWR__APPLSTARTUP);

  // Connect to database
  sts = gdh_Init( "ra_myappl");
  if ( EVEN(sts)) throw co_error(sts);

  // Register application object
  sts = gdh_NameToObjid( "Nodes-Saturnus7-ra_myappl", &aoid);
  if ( EVEN(sts)) throw co_error(sts);

  aproc_RegisterObject( aoid);

  // Directlink to curve object
  sts = gdh_RefObjectInfo( name, (void **)&curve_ptr, &dlid, sizeof(*curve_ptr));
  if ( EVEN(sts)) throw co_error(sts);

  errh_SetStatus( PWR__ARUN);
}

void ra_myappl::scan()
{
  for ( unsigned int i = 0;;i++) {
    // Notify that we are still alive
    aproc_TimeStamp();
```

```
    if ( i % 5 == 0) {
      for ( int j = 0; j < 100; j++) {
        curve_ptr->XValue[j] = j;
        curve_ptr->YValue[j] = 50 + 50 * sin( 2.0 *  M_PI * (j + i) / 100);
      }
      curve_ptr->Update = 1;
    }
    else if ( i % 5 == 2)
      curve_ptr->Update = 0;
    sleep(1);
    if ( i > 360)
      i = 0;
  }
}

void ra_myappl::close()
{
  gdh_UnrefObjectInfo( dlid);
}


int main()
{
  ra_myappl myappl;

  try {
    myappl.init();
  }
  catch ( co_error e) {
    errh_Fatal( "ra_myappl terminated, %m", e.sts());
    errh_SetStatus( PWR__APPLTERM);
    exit(0);
  }
  myappl.scan();
  myappl.close();
}
```

16.3 Start av applikation

En applikation som ska startas vid uppstart av ProviewR runtime läggs in i applikationsfilen. Det ligger på $pwrp_load och har namnet ld_appl_'nodename'_'qbus'.txt, t ex

```
$pwrp_load/ld_appl_mynode_999.txt
```

I filen lägger man en rad för varje applikation som ska startas

```
# id       name      [no]load [no]run file       prio  [no]debug  "arg"
ra_myappl, ra_myappl, noload,  run,   ra_myappl, 12,   nodebug,   ""
```

16.4 Ta emot systemhändelser

ProviewR skickar ut medelanden vi olika händelser, t ex, när en mjuk omstart pågår eller om runtimmiljön stoppas. En applikation kan avlyssna dessa händelser, för att t ex terminera när ProviewR stoppas. Avlyssningen sker via Qcom. Man skapar en Qcom kö, och knyter denna kö till den kö som meddelandena sänds ut på.

```
#include "rt_qcom.h"
#include "rt_ini_event.h"
#include "rt_qcom_msg.h"

qcom_sQid qid = qcom_cNQid;
qcom_sQid qini;
qcom_sQattr qAttr;

if ( !qcom_Init(&sts, 0, "ra_myappl")) {
  throw co_error(sts);

// Create a queue to receive stop and restart events
qAttr.type = qcom_eQtype_private;
qAttr.quota = 100;
if ( !qcom_CreateQ(&sts, &qid, &qAttr, "events"))
  throw co_error(sts);

// Bind to init event queue
qini = qcom_cQini;
if ( !qcom_Bind(&sts, qid, &qini))
  throw co_error(sts);
```

I varje scan läser man av kön med qcom_Get() för att se om något meddelande har kommit. Man kan även passa på att använda timeouten i qcom_Get() för att vänta till nästa scan.
I exemplet nedan hanterar man dels terminate meddelandet, men även oldPlcStop och swapDone som markerar början och slutet på en mjuk omstart. Det behöver man enbart göra om man vill att applikationen ska kunna upptäcka nya objekt eller konfigureringar vid en mjuk omstart.

```
int tmo = 1000;
char mp[2000];
qcom_sGet get;
int swap = 0;

for (;;) {
  get.maxSize = sizeof(mp);
  get.data = mp;
  qcom_Get( &sts, &qid, &get, tmo);
  if (sts == QCOM__TMO || sts == QCOM__QEMPTY) {
    if ( !swap)
      // Do the normal thing
      scan();
  }
  else {
    // Ini event received
    ini_mEvent  new_event;
    qcom_sEvent *ep = (qcom_sEvent*) get.data;
```

```
    new_event.m  = ep->mask;
    if (new_event.b.oldPlcStop && !swap) {
      errh_SetStatus( PWR__APPLRESTART);
      swap = 1;
      close();
    } else if (new_event.b.swapDone && swap) {
      swap = 0;
      open();
      errh_SetStatus( PWR__ARUN);
    } else if (new_event.b.terminate) {
      exit(0);
    }
  }
}
```

Om man endast är intresserad av att stoppa processen när ProviewR tas ner, finns det ett enklare sätt att ta död på den. Man kan lägga en scriptfile pwrp_stop.sh på $pwrp_exe där man gör kill på processen.

```
killall ra_myappl
```

16.5 Basklass för applikationer rt_appl

Basklassen rt_appl innehåller mycket av de initieringar och övervakning av händelser som är beskrivet ovan. Genom att subklassa rt_appl behöver man inte lägga in någon kod för detta utan det utförs av rt_appl. rt_appl innehåller tre virtuella funktioner som ska implementeras av subklassen, open(), close() och scan(). open() används vid initieringen för att direktlänka mot attribut och objekt, scan() blir anropad cykliskt med angiven scantid, och i close() tar man ner direktlänkningarna.

rt_appl gör följande:

- Initiera gdh, errh och qcom.
- Sätter applikationsstatus vid uppstart och omstart.
- Hanterar händelser för mjuk omstart och terminiering.
- Tidstämplar för undvika timeout.

I exemplet nedan kan vi se applikationen ra_appl som subklassar rt_appl.

```
class ra_appl : public rt_appl {
public:
 ra_appl() : rt_appl( "ra_appl", errh_eAnix_appl1) {}
 void open();
 void close();
 void scan();
};

void ra_appl::open()
{
 // Link to database objects
}
```

```
void ra_appl::close()
{
 // Unlink to database objects
}

void ra_appl::scan()
{
 // Do something
}

int main()
{
 ra_appl appl;

 appl.init();
 appl.register_appl( "Nodes-MyNode-MyAppl");

 appl.mainloop();
}
```

16.6 Skicka larm och meddelanden

Från en applikation kan man även skicka larm och meddelanden till operatörens larm och händelselista. Först måste man knyta upp sig mot eventmonitorn med mh_ApplConnect(). Vi skickar med objektidentiteten för applikationsobjektet som första argument. För subclasser till rt_appl can identiteten hämtas med apploid().

```
#include "rt_mh_appl.h"

pwr_tUInt32 num;
pwr_tOid aoid = apploid();
sts = mh_ApplConnect( aoid, mh_mApplFlags(0), "", mh_eEvent_Info, mh_eEventPrio_A,
   mh_mEventFlags_Bell, "", &num);
  if (EVEN(sts)) throw co_error(sts);
```

Därefter kan vi skicka larm med mh_ApplMessage().

```
mh_sApplMessage msg;
pwr_tUInt32 msgid;

memset( &msg, 0, sizeof(msg));
msg.EventFlags = mh_mEventFlags(mh_mEventFlags_Returned |
      mh_mEventFlags_NoObject |
      mh_mEventFlags_Bell);
time_GetTime( &msg.EventTime);
strcpy( msg.EventName, "Message from ra_myappl");
strcpy( msg.EventText, "I'm up and running now !");
msg.EventType = mh_eEvent_Alarm;
msg.EventPrio = mh_eEventPrio_B;
```

```
sts = mh_ApplMessage( &msgid, &msg);
if (EVEN(sts)) throw co_error(sts);
```

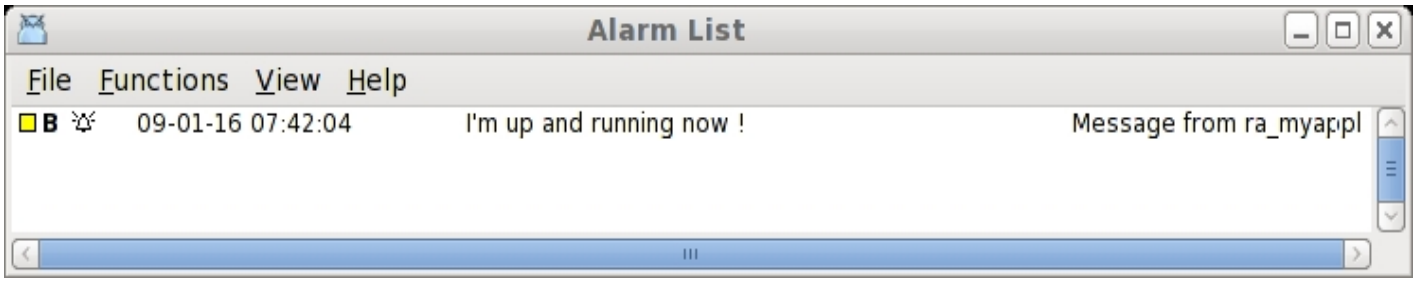


Fig Larmet i larmlistan

16.7 Kommunicera med andra processer

ProviewR's protokoll för kommunikation mellan processer kan med fördel användas även av applikationer. Kommunikation kan ske

- mellan processer inom samma nod.
- mellan processer på olika noder som tillhör samma projekt.
- mellan processer på noder som tillhör olika projekt, om projekten tillhör samma Qcom bus. I det här fallet måste noderna konfigureras med FriendNodeConfig objekt där Connection sätt till QcomOnly.

Läs mer om Qcom i Qcom Reference Guide.

16.8 Hämta data från lagringsstation

Data som finns lagrat på en ProviewR lagringsstation kan hämtas med klient gränssnittet sevcli. Först initierar man sevcli med sevcli_init() och anger vilken lagringsstation man ska hämta data ifrån med sevcli_set_servernode().

```
sevcli_tCtx sevctx;
char server_node[40] = "MyStorageStation";

if ( !sevcli_init( &sts, &sevctx))
  throw co_error(sts);

if ( !sevcli_set_servernode( &sts, sevctx, server_node))
  throw co_error(sts);
```

Därefter kan man hämta data med sevcli_get_itemdata(). Data identifieras med objektsidentitet och attribut, dessutom anger man tidsintervall för det data som ska hämtas, och max antal punkter.

```
pwr_tTime *time_buf;
void *value_buf;
pwr_tTime from = pwr_cNTime;
pwr_tTime to = pwr_cNTime;
int rows;
pwr_eType vtype;
```

```
unsigned int vsize;
pwr_tOName name = "H1-H2-Temperature";
pwr_tOName aname = "ActualValue";
pwr_tObjid oid;
char timstr[40];

sts = gdh_NameToObjid( name, &oid);
if (EVEN(sts)) throw co_error(sts);

if ( !sevcli_get_itemdata( &sts, sevctx, oid, aname, from, to, 1000, &time_buf, &value
                           &rows, &vtype, &vsize))
  throw co_error(sts);

for ( int i = 0; i < rows; i++) {
  time_AtoAscii( &time_buf[i], time_eFormat_DateAndTime, timstr, sizeof(timstr));

  cout << timstr << " " << ((pwr_tFloat32 *)value_buf)[i] << endl;
}

free( time_buf);
free( value_buf);
```

Slutligen anropar man sevcli_close() för att koppla ner anslutningen mot servernoden.

```
sevcli_close( &sts, sevctx);
```

16.9 I/O hantering

Har man stora krav på snabbhet och synkronisering kan en applikation arbeta direkt mot I/O systemet och själv läsa in och ställa ut data.

Initieringen sker med funktionen io_init() till vilken man bl a skickar argumentet process. Process identifierar vilka io-enheter (agent, rack eller kort) som ska hanteras av en viss process. Varje io-objekt har ett Process attribut som är en bitmask, och om detta överensstämmer med det som skickas som argument till io_init, kommer enheten att hanteras av applikationen. Om ett kort hanteras av en applikation måste även det rack och den agent som kortet tillhör, hanteras av applikationen.

Eftersom Process-attributet är en bitmask kan en enhet hanteras av flera processer, genom att flera bitar sätts. Om man t ex har flera kort i ett rack, och vissa kort hanteras av plc-processen och vissa av en applikation, måste rack-enheten hanteras av både plc't och applikationen. Hur och om det fungerar att låta en enhet hanteras av flera processer beror på hur io-metoderna är skrivna. För Profibus t ex, kan man inte dela upp hanteringen av slavar på olika processer.

```
#include rt_io_base.h

io_tCtx io_ctx;

sts = io_init( io_mProcess_User, pwr_cNOid, &io_ctx, 0, scantime);
if ( EVEN(sts)) {
  errh_Error( "Io init error: %m", sts);
  throw co_error(sts);
```

```
}
```

Läsning sker med io_read() som läser data från io-enheterna och lägger ut i värdena i de signaler som hör till enheten. Lämpligtvis direktlänkar sig applikationen mot dessa signaler och även mot signalerna för utenheterna som skrivs med funktionen io_write().

```
sts = io_read( io_ctx);

sts = io_write( io_ctx);
```

16.10 Trådsäkra tider och strängar

Tid- och strängattribut i realtidsdatabasen kan inte läsas eller skrivas i en atomisk operation, och behöver därför speciell hänsyn vid använding om attributet används i flera olika trådar eller processer. Problemet är att medan en tråd skriver ett tids eller strängattribut, kan en annan tråd läsa attributet innan skrivninget är avslutat, och därmed läsa ett värde som innehåller delar av både det gamla och det nya värdet. För att unvika detta ska läs och skrivoperationer skyddas av ett lås. Det finns två lås, ett för tidsattribut och ett för strängattribut.

Tidslås

Tidslåset används för attribut av typerna pwr_tTime och pwr_tDeltaTime. Varje gång ett tidsattribut läses eller skrivs ska låset sättas. Följande plc funktionsobjekt använder låset för att skydda läsning och skrivning av tidsvärden.
StoATv, StoDTv, CStoATv, CStoDTv, StoDTp, StoATp, CStoDTp, CStoATp, GetATp, GetDTp, GetATv, GetDTv, CurrentTime.

Notera att andra funktionsobjekt inte är skyddade av låset, Det är inte trådsäkert att använda utgången från t ex en AtAdd i en annan plctråd eller applikationsprogram. I det här fallet ska tiden först lagras i ett ATv-objekt.

Applikationer ska använda följande gdh-funktioner för att läsa och skriva tider

```
void gdh_GetTimeDL( pwr_tTime *atp, pwr_tTime *time);
void gdh_SetTimeDL( pwr_tTime *atp, pwr_tTime *time);
void gdh_GetDeltaTimeDL( pwr_tDeltaTime *dtp, pwr_tDeltaTime *time);
void gdh_SetDeltaTimeDL( pwr_tDeltaTime *dtp, pwr_tDeltaTime *time);
pwr_tStatus gdh_GetObjectInfoTime( char *name, pwr_tTime *time);
pwr_tStatus gdh_SetObjectInfoTime( char *name, pwr_tTime *time);
pwr_tStatus gdh_GetObjectInfoDeltaTime( char *name, pwr_tDeltaTime *time);
pwr_tStatus gdh_SetObjectInfoDeltaTime( char *name, pwr_tDeltaTime *time);
```

Innan någan av dessa funktioner används, ska tidslåset initieras av applikationen med ett anrop till lck_Create(), t ex

```
lck_Create(&sts, lck_eLock_Time);
```

Stränglås

Stränglåset används för strängattribut. Följande plc-funktionsobjekt använder låset.
StoSv, CStoSv, StoSp, CStoSp, StoNumSp, CStoNumSp, GetSp, GetSv.

Andra funktionsobjekt är inte skyddade av låset och strängvärden från dessa objekt ska inte läsas eller skrivas från andra plc-trådar eller i applikationsprogram.

Applikationer ska använda dessa gdh-funktioner för att läsa och skriva strängattribut

```
void gdh_GetStrDL( char *sp, char *str, int size);
void gdh_SetStrDL( char *sp, char *str, int size);
pwr_tStatus gdh_GetObjectInfoStr( char *name, char *str, int size);
pwr_tStatus gdh_SetObjectInfoStr( char *name, char *str, int size);
```

Innan någon av dessa funktioner används, ska stränglåset initieras av applikationen med ett anrop till lck_Create(), t ex

```
lck_Create(&sts, lck_eLock_Str);
```

16.11 Bygga en applikation

En c++ applikation måste kompileras och länkas, och man kan använda make för detta. ProviewR har en regelfil $pwr_exe/pwrp_rules.mk som bl a innehåller regler för kompilering.

En make fil för applikationen ra_myappl på katalogen $pwrp_src/myappl kan se ut på så här ($pwrp_src/myappl/makefile):

```
ra_myappl_top : ra_myappl

include $(pwr_exe)/pwrp_rules.mk

ra_myappl_modules : \
  $(pwrp_obj)/ra_myappl.o \
  $(pwrp_exe)/ra_myappl

ra_myappl : ra_myappl_modules
 @ echo "ra_myappl built"

#
# Modules
#

$(pwrp_obj)/ra_myappl.o : $(pwrp_src)/myappl/ra_myappl.cpp \
                          $(pwrp_src)/myappl/ra_myappl.h

$(pwrp_exe)/ra_myappl : $(pwrp_obj)/ra_myappl.o
 @ echo "Link $(tname)"
 @ $(ldxx) $(linkflags) -o $(target) $(source) -lpwr_rt -lpwr_co \
  -lpwr_msg_dummy -lrpcsvc -lpthread -lm -lrt
```

Makefilen kan exekveras genom att man ställer sig på den aktuella katalogen och skriver make

```
make
```

Man kan även lägga in byggkommandot i Application objektet för applikationen i attributet BuildCmd. I det här fallet blir byggkommandot

```
make --directory $pwrp_src/myappl -f makefile
```

Det här kommandot exekveras då i samband med att noden byggs från konfiguratören. På det här sättet kan man försäkra sig om att alla applikationer uppdateras när noden byggs om.

16.12 Java applikationer

En del API finns även för java i form av klasserna Gdh, Errh och Qcom. Här följer en exempel på en java applikation som knyter upp sig mot realtidsdatabasen och läser resp skriver ett värde i den.

```
import jpwr.rt.*;

public class MyJappl {
  public MyJappl() {
    Gdh gdh = new Gdh( null);

    CdhrBoolean rb = gdh.getObjectInfoBoolean( "H1-H2-Start.ActualValue");

    PwrtStatus rsts = gdh.setObjectInfo( "H1-H1-Start.ActualValue",
         !rb.value);
  }

  //Main method
  public static void main(String[] args) {
    new MyJappl();
  }
}
```

För att kompilera och exekvera måste man lägga in $pwr_lib/pwr_rt.jar och arbetskatalogen i CLASSPATH, samt $pwr_exe i LD_LIBRARY_PATH

```
> export CLASSPATH=$pwr_lib/pwr_rt.jar:$pwrp_src/myjappl
> export LD_LIBRARY_PATH=$pwr_exe
```

Kompilera med
```
> javac MyJappl.java
```

och kör med
```
> java MyJappl
```

För auto-start skapar man ett script som exporterar CLASSPATH och LD_LIBRARY_PATH, och starar java-applikationen. Scriptet läggs in i appl-filen på samma sätt som en c-applikation.

17 Skapa processbilder

Det här kapitlet beskriver hur man skapar processbilder.

Processbilder ritas och konfigureras i Ge editorn.

Ge editorn

Ge öppnas från menyn i konfiguratören, 'Functions/Open Ge'. Den består av

- en verktygspanel.
- en arbetsarea.
- en subgraf palett.
- en färg palett.
- ett fönster som visa anläggningshierakin.
- ett navigationsfönster.

Bakgrundsbild

En bakgrundsbild ritas med basobjekt som rektanglar, cirklar, linjer, polylinjer och texter. Dessa hittar man i verktygspanelen. Skapa att basobjekt genom att aktivera tryckknappen i verktygspanelen, och dra eller klicka med MB1 i arbetsarean. Om basobjektet ska vara fyllt, väljer man objektet och aktiverar 'Fill' i vertygspanelen. Man ändar fyllnadsfärg genom att välja ut objektet och klicka på den önskade färgen i färgpaletten. Kantfärg ändras genom att man klickar med MB2 i färgpaletten, och textfärg med Shift/Klick MB1.

Subgrafer

En subgraf är en komponent, t ex en ventil, en motor eller en tryckknapp. För att skapa en subgraf väljer man ut subgrafen i subgraf paletten, och klickar med MB2 i arbetsarean.

Grupper

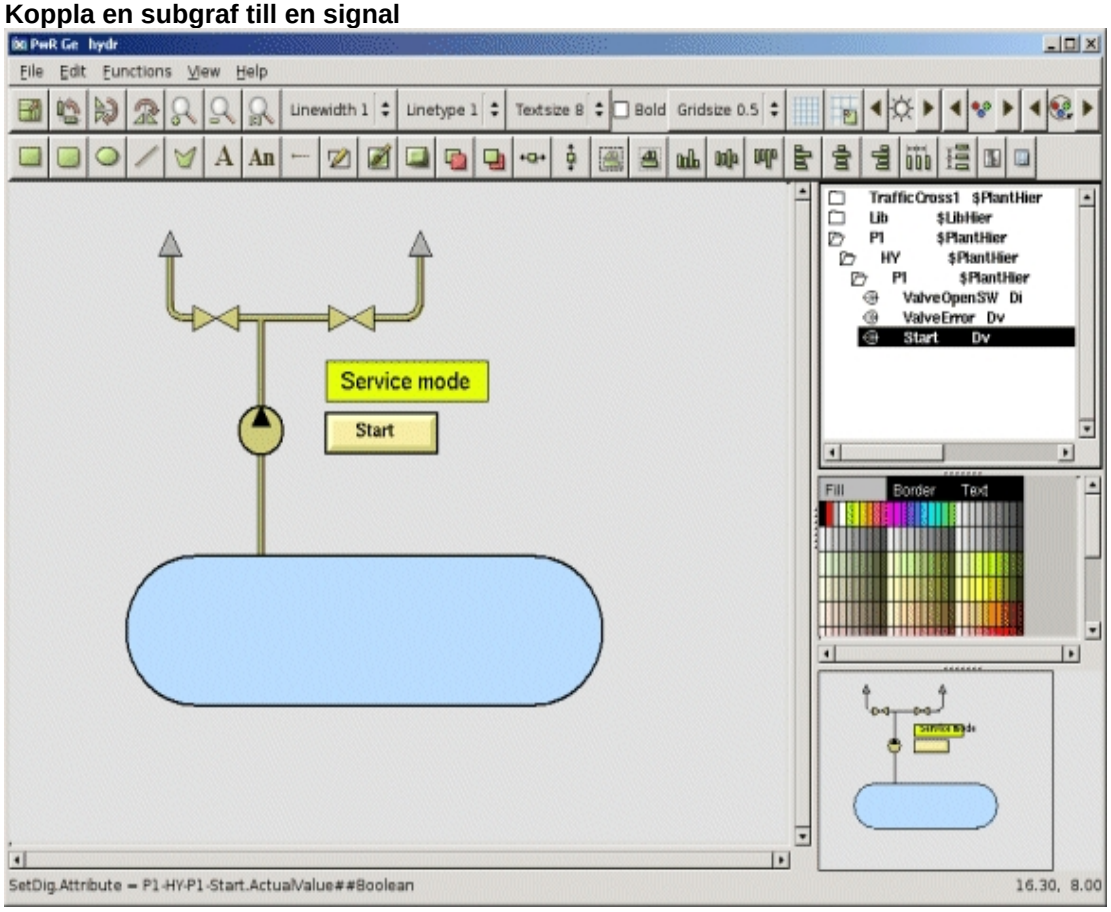
Basobjekt och subgrafsobjekt kan grupperas genom att välja ut dem, och aktivera 'Functions/Group' i menyn.

Dynamik

Subgrafer och grupper har dynamiska egenskaper, dvs de kan kopplas till en signal i runtime databasen, och ändra färg, position eller form beroende på signalernas värde. En subgraf har ofta en default dynamik, t ex en indikator kan skifta mellan två färger. Man behöver endast koppla ventilen till en digital signal för att den ska fungera. Det här görs genom att välja ut signalen i anläggningshierakin, och klicka på ventilen med Ctrl/dubbelklick MB1.

En tryckknapp har aktions egenskaper, den sätter, återställer eller togglar en signal i databasen. En knapp med sätt-aktion skapas genom att välja en ButtonSet i subgraf paletten under mappen Pushbuttons, och klicka med MB2 i arbetsaran. Den signal som ska sättas kopplas som tidigare genom att välja ut signalen och klicka med Ctrl/dubbelklick MB1 på tryckknappen.

Koppla en subgraf till en signal



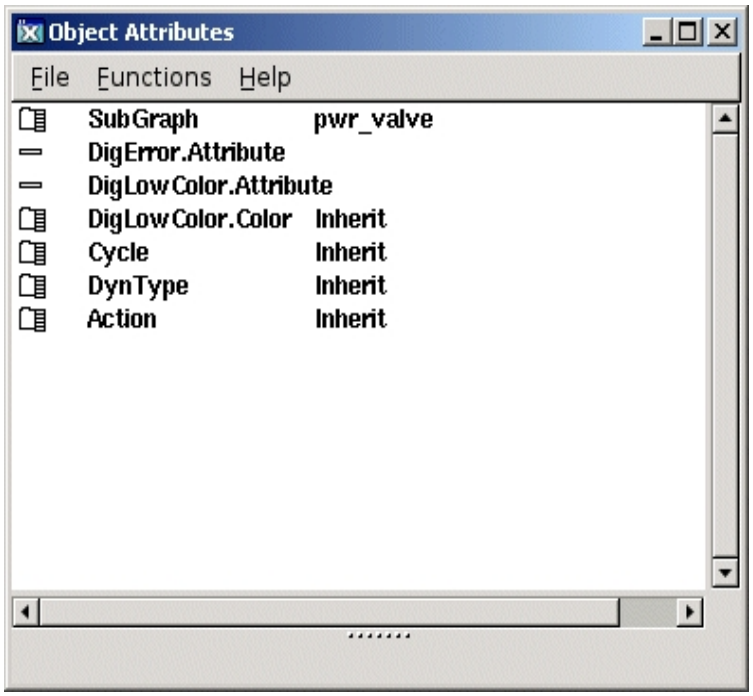
Även grupper har dynamik, dvs de kan skifta färg, ändra position, eller utföra någon aktion. De har inte någon default dynamik eller defaultfärg som subgraferna, utan detta måste anges för varje grupp

Attribut editor

Basobjekt, subgrafsobjekt och grupper har olika attribut, som man påverkar från objekteditorn. Objekteditorn öppnas genom att man väljer ut objektet, och aktivera 'Function/Object attributes' i menyn. Genom att ta upp attributseditorn för tryckknappen ovan, kan man t ex mata in en text som visas på tryckknappen i attributet 'Text'.

Om en subgraf har med avancerad dynamik, t ex växla mellan flera färger, så måste man ofta koppla den till flera signaler. Om man t ex, öppnar men attributeditorn för en ventil, ser man att den kan kopplas till två attribut, 'DigError.Attribute' och 'DigLowColor.Attribute'. DigError attributet markera att någonting är fel, och om denna signal är sann, färgas ventilen röd. DigLowColor attributet kopplas till ventilens gränsläge. Om denna signal är falsk, färgas ventilen i den färg som anges i 'DigLowColor.Color'. Är signalen sann, behåller ventilen den färg som den ges i editorn. Signalerna för de två attributen läggs in genom att man väljer ut respektive attribut i anläggningshierakin, och klickar med Ctrl/Dubbelklick MB1 på attributraden i attributeditorn. Färgen 'DigLowColor' anges genom att öppna attributet och välja ut en av de 300 färgerna. Färgerna ligger i samma ordning som i färgpaletten och med lite övning identifieras med hjälp av namnet.

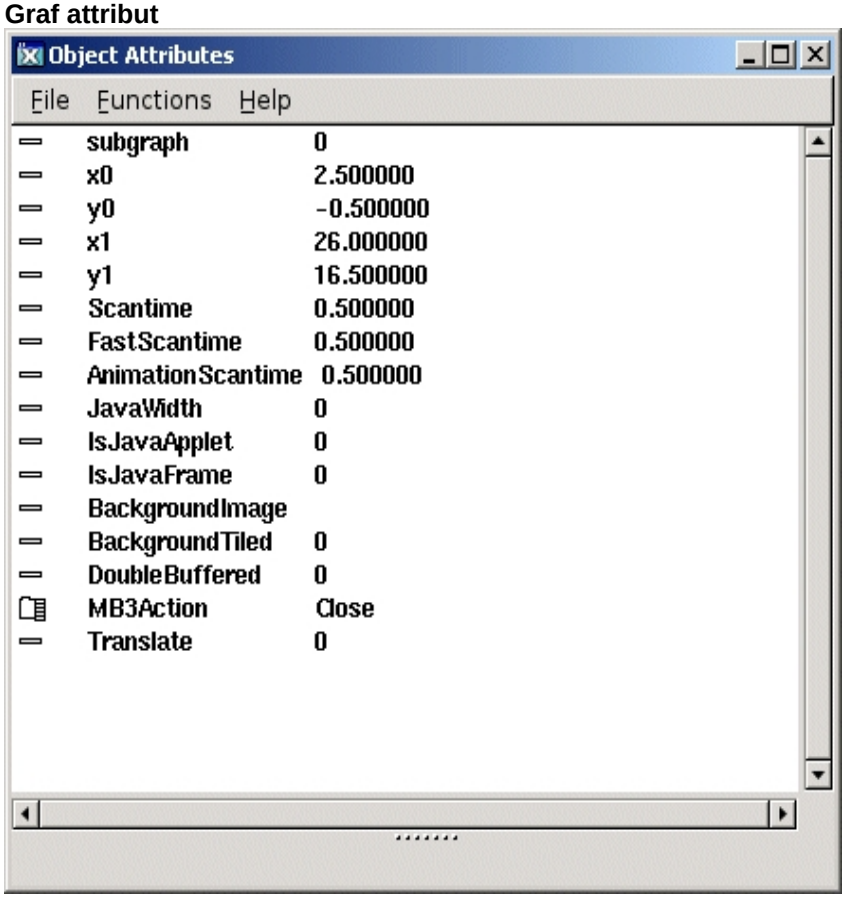
Attributeditorn för en ventil



Bildområde

Ritarean i Ge är obegränsad i alla riktningar, så innan man sparar bilden, ska man ange gränserna för bilden i x och y-led. Öppna graf attributen med 'File/Graph attributes' i menyn. Mät upp koordinaterna för övre vänstra hörnet, och mata in dem i x0 och y0, mät sedan upp koordinaterna för nedre högra hörnet och mata in dem i x1 och y1. Mätningen görs genom att placera makören på positionen och läsa av koordinaterna på informations raden.

Graf attribut



Konfigurering i arbetsbänken

XttGraph objektet

Till varje processbild hör ett XttGraph objekt. Det här objektet är vanligtvis ett barn till operatörsplats objektet (OpPlace) för noden, på vilken bilden ska visas. Man behöver ett XttGraph objekt på varje node, som bilden ska visas på. Däremot behöver man enbart en graf fil. Följande attribut i XttGraph objektet ska ges lämpliga värden:

- Action, namnet på pwg filen, inklusive filtyp, t ex 'hydr.pwg'.
- Title, titel på bildfönstret.
- ButtonText, text på tryckknappen i grafikfönstret.

XttGraph objektet innehåller andra attribut som t ex bestämmer bildfönstrets storlek och läge på skärmen. Dessa attribut beskrivs i detalj i objektshanboken.
Se XttGraph i Objektshandboken

18 Web operatörsmiljö

Utöver den vanliga operatörsmiljön i X windows, innehåller ProivewR också en operatörsmiljö skriven i HTML5 och javascript. Miljön är körbar på de flesta webläsare på telefoner, surfplattor och persondatorer.

Gränssnittet består av tre delar
- ett operatörsfönster som öppnas i webläsaren.
- en server process som förser operatörsfönstret med information via en websocket.
- en katalog där filer är tillgängliga för webläsaren.

Operatörsfönster

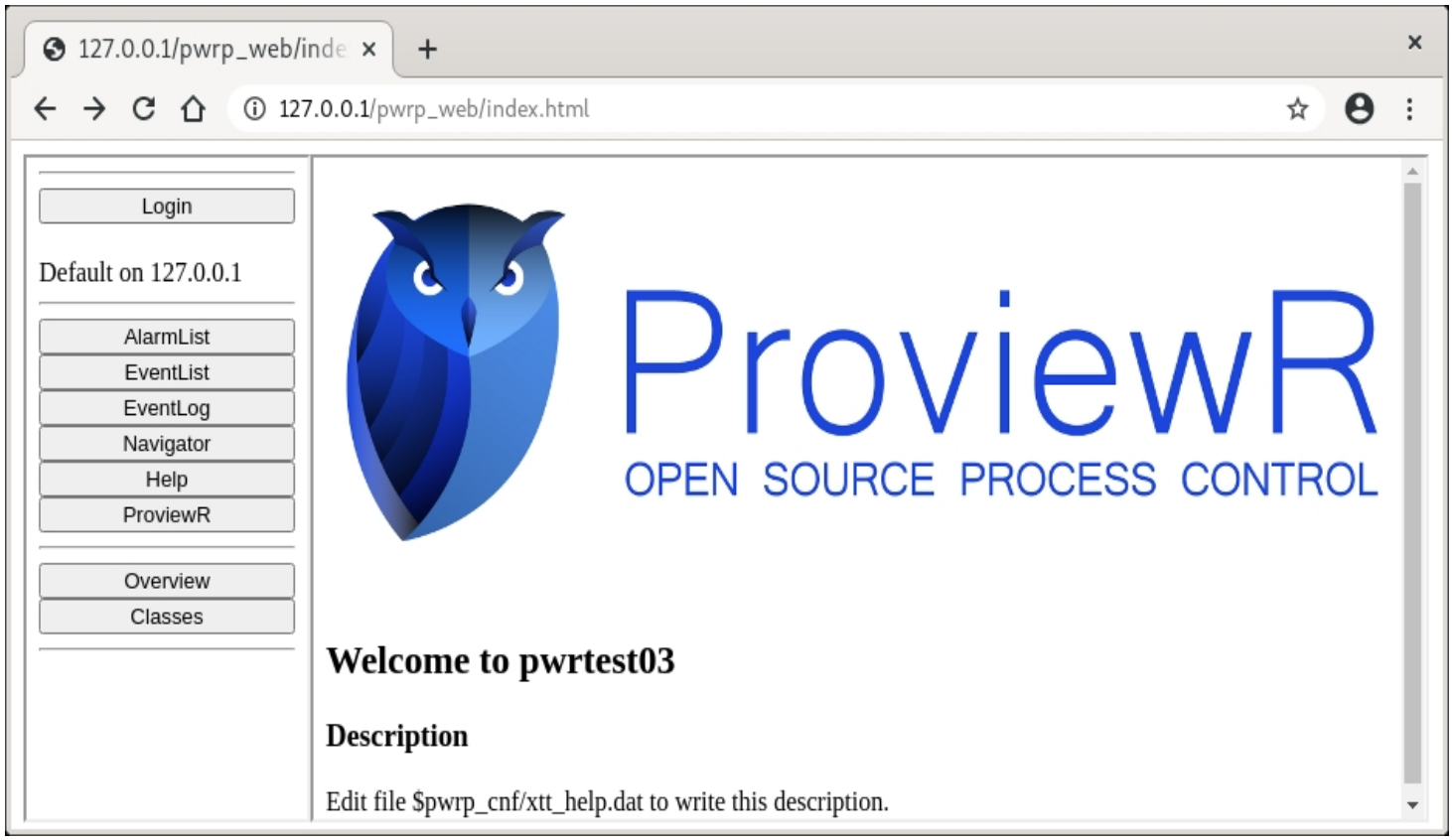


Fig Operatörsfönstret

Operatörsfönstret konfigureras med ett OpPlaceWeb-objekt i nodhierarkin. Förstret innehåller en meny till vänster och ett startsida till höger.

Menyn är indelad i tre sektioner. Den första innehåller att antal standardfunktioner
- val av språk.
- in och utloggning.
- öppna larm och händelselistor.
- öppna händlseloggen.
- öppna runtime-navigatorn.

- länk till projektets hjälptext.
- länk till dokumentationen för ProviewR.

Om de här knapparna ska visas eller inte kan konfigureras med Enable/Disable attributen i OpPlaceWeb-objektet.

Den andra sektionen innehåller knappar för att öppna Ge grafer konfigurerade med WebGraph-objekt. WebGraph-objekten är placerade som barn till OpPlace-objektet och innehåller namnet på Ge grafens pwg-fil.

Den tredje sektionen innehåller länkar definierade med WebLink-objekt. WebLink-objekten innehåller en URL och är placerade under OpPlaceWeb-objektet.

Den högra sidan innehåller hjältexten för projektet, skriven i filen $pwrp_cnf/xtt_help.dat.
Om någon annan sida ska visas, kan den anges i StartURL attributet i OpPlaceWeb objektet.

Det kan finnas flera OpPlaceWeb-objekt på en nod, som konfigurerar websidor med olika layout. OpPlaceWeb-objeken ska ha olika filnamn angivna i FileName-attributet, och kan öppnas med en URL till den specificerade filen. Default-filen är index.html för det första OpPlaceWeb-objektet och index2.html etc för de nästkommande OpPlaceWeb objekten.

Opertörsplatsen med FileName index.html kan öppnas med URL'en

```
http://'hostname'/pwrp_web/index.html
```

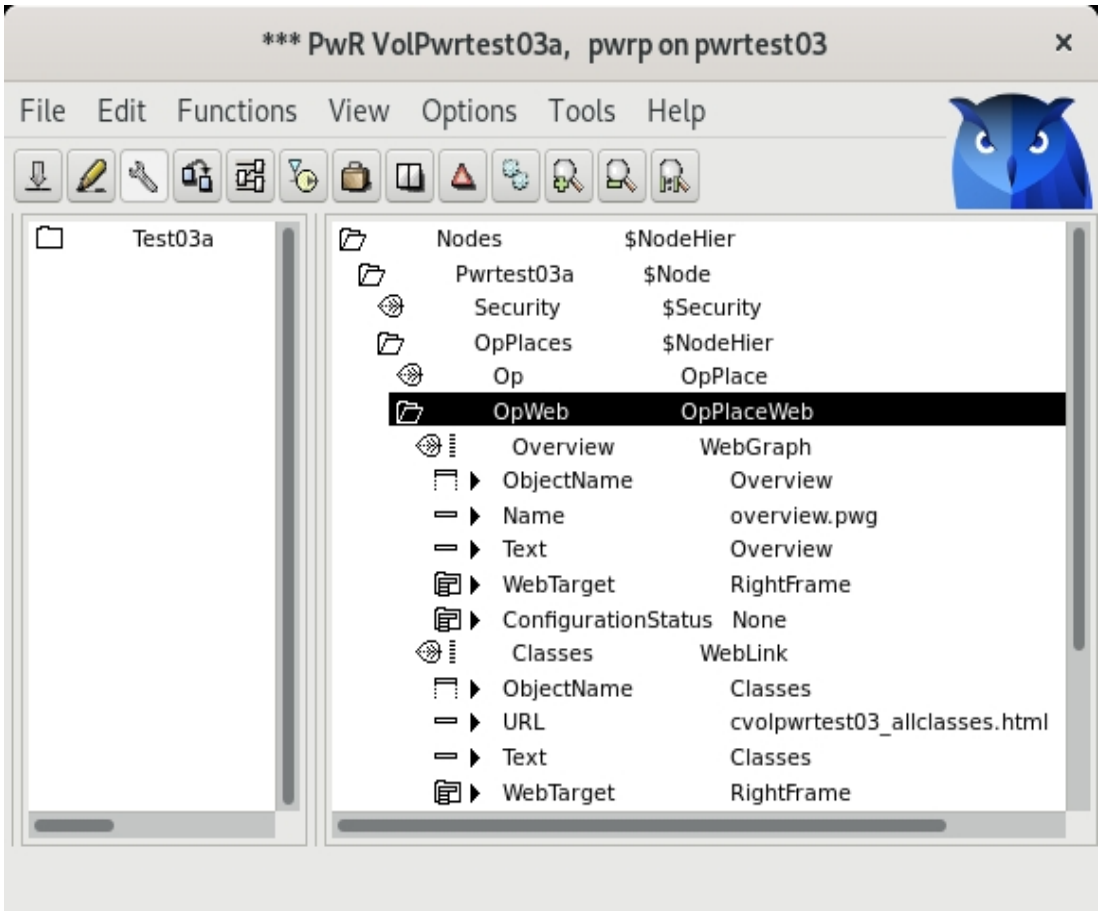


Fig Konfigurering av weboperatör

Serverprocess

Serverprocessen konfigureras med ett WebSocketServer-objekt i nodhierarkin. Servern är en java process och kräver att java är installerat. Servern lyssnar efter förfrågningar från operatörsfönster att öppna en websocket, och tillhandahåller information om realtidsdatabasen till operatörsfönstret.

Larm och händelselista

För larm och händelselistorna måste EventSelectList-attributet i WebSockerServer-objektet fyllas i med hierarkier från vilka larm och händelser ska visas.

Web katalogen

Operatörsfönstret omfattar ett antal filer, och dessa samlas på web-katalogen, pwrp_web. Katalogen registreras i webservern och görs tillgänglig från webben.

Här listas några filer som ska finnas på katalogen

- pwg-filer for Ge grafer som används i WebGraph-objekt. Ska kopieras från $pwrp_pop.
- pwg-filer för objektsbilder, kan kopieras från $pwr_exe.
- flw-filer för plc trace, ska kopieras från $pwrp_load.
- korsreferensfiler, rtt_crr*.dat ska kopieras från $pwrp_load.
- html-filer för xtt hjälptexter. Genereras när noden byggs.
- dokumentation av lokala klasser genereras när klassvolymen byggs.
- Alla filer på $pwr_web kopieras till $pwrp_web vid installation av pwrrt-paketet.

Vissa filer genereras eller kopieras vid installationen, men andra måste kopieras manuellt eller med automatik genom att skapa Build-objekt i direktory-volymen. I exemplet nedan visas BuildCopy objekt för att kopiera

- pwg-filer från $pwrp_pop.
- flw-filer från $pwrp_load.
- korsreferensfiler från $pwrp_load.

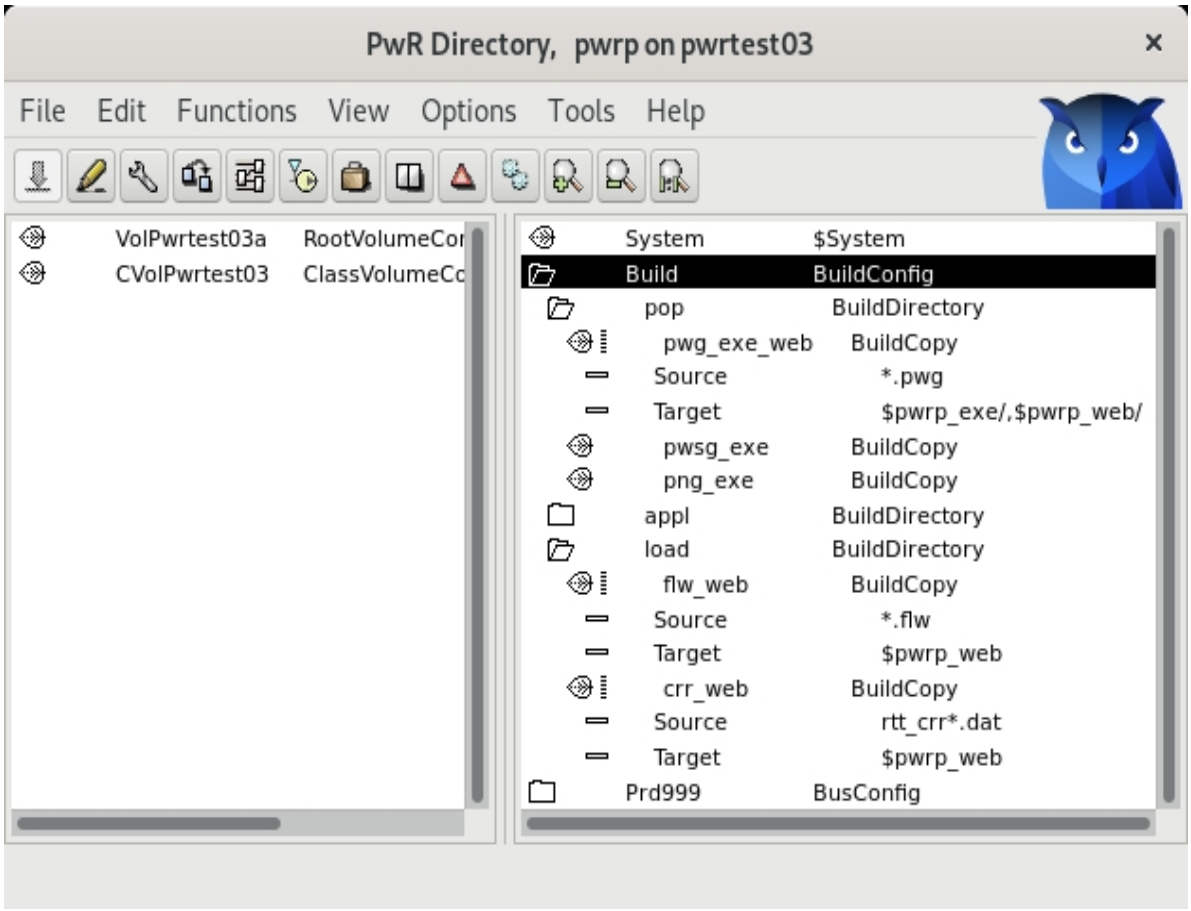


Fig Build konfigurering för pwrp_web-katalogen

Hjälptexter

Hjälptexterna för projektet i filen xtt_help.dat konverteras till html-filer på $pwrp_web av bygg-metoden för OpPlaceWeb objektet. Startsidan för hjälptexterna är $pwrp_web/xtt_help_index.html. Denna visas som default till höger i operatörsfönstret om inte någon annan URL är angiven in StartURL-attributet i OpPlaceWeb-objektet.

Om hjälptexen innehåller image taggar, måste png eller gif-filerna för dessa kopieras till $pwrp_web.

Distribution

Filerna på $pwrp_web ska ingå i distributionspaketet, och distribueras till $pwrp_web på process eller operatörsstationen. Det görs genom att sätta WebFiles i Distributions-objektet under NodeConfig-objektet i directoryvolymen.

Konfigurering av webservern

$pwrp_web katalogen ska vara tillgänglig från webben, och läggas in i webserverns konfigurering. För nginx och apache gör detta vid installeringen av pwrrt-paketet, men om projektet t ex ska visas från utvecklingsstationen måste det göras manuellt. Här följer några exempel.

nginx

Lägg till de här raderna i /etc/nginx/sites-enabled/default (byt /pwrp/common/web/ mot den verkliga katalogen för $pwrp_web)

```
location /pwrp_web/ {
  alias /pwrp/common/web/;
}
```

apache
Lägg till de här raderna i /etc/apache2/apache2.conf (byt /pwrp/common/web/ mot den verkliga katalogen för $pwrp_web)

```
Alias /pwrp_web/ /pwrp/common/web/

<Directory /pwrp/common/web>
    AllowOverride None
    Require all granted
</Directory>
```

Auktorisering

Om inloggnings-funktionen är aktiverad, kan en användare logga in med ett giltigt namn och passerord, och erhålla de privilegier som tillhör den inloggade användaren. Auktoriserade användare är medlemmar av den systemgrupp som är angiven i WebSystemGroup i Security-objektet.

I exemplet nedan, används systemgruppen 'Common' som innehåller de ordinare användaren pwrp, system etc, men rekommandationen är att skapa en specifik systemgrupp med särskilda användare för webben.

För ej inloggade användare bestäms privilegierna av DefaultWebPriv i Security-objektet. Det rekommenderas att DefaultWebPriv är RtRead eller noll (inga privilegier). Användare utan privilegier kan endast aktivera login-funktionen.

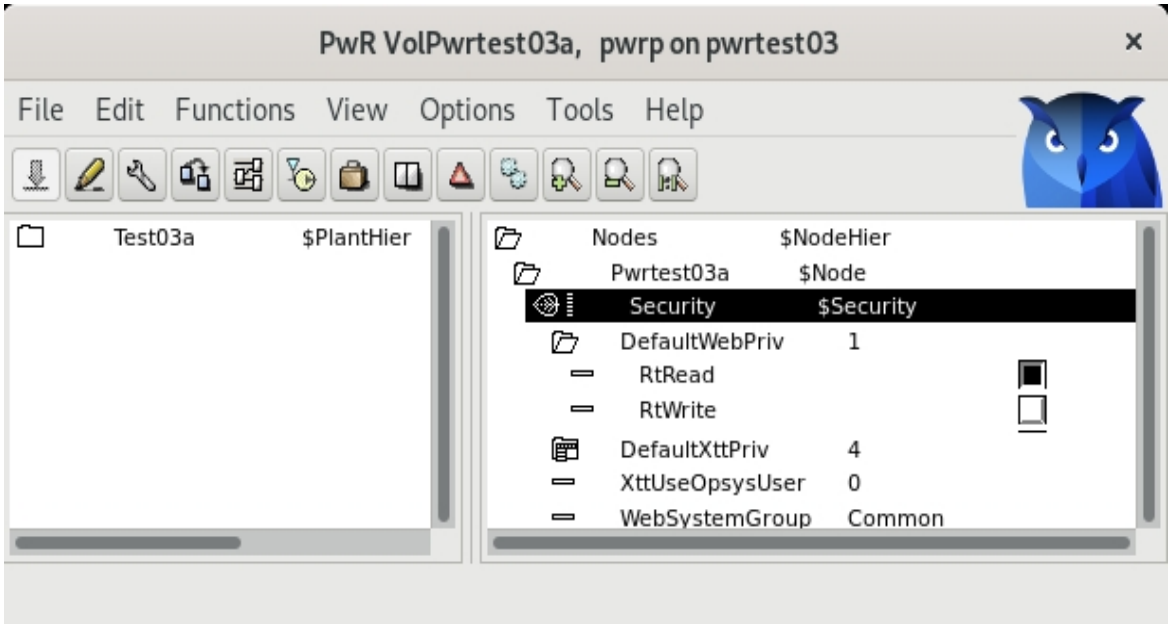


Fig Konfiguration av auktorisering

19 Starta och testa ett ProviewR system

I föregående kapitel har vi beskrivit hur man konfigurerar ett ProviewR system, hur man skapar PLC program och processbilder. Nu är det dags att köra och testa systemet.

I avsnitten visas hur man

- bygger systemet.
- startar simulerings miljö
  och runtime monitorn.
- distribuerar.
- startar runtime miljö.

Syntax kontroll

Vissa klasser har en syntax metod som kontrollerar att objektet är rätt konfigurerat. Metoden för signaler kontrollerar t ex att de är kopplade till en kanal, metoden för ett PlcPgm att den är kopplade till ett trådobjekt etc.
Syntax metoderna fångar upp en del fel, men är inte någon garanti för allt fungerar felfritt.

Syntax-metoderna körs när 'Functions/Syntax Check' aktiveras i Configuratorn.

19.1 Bygga

Innan man har en körbart system för en nod måste man bygga noden. Det innebär att man genererar alla de filer som behövs i runtime-miljön, t ex en boot-fil som talar om vilka volymer som ska laddas, laddatafiler för volymerna som innehåller info om objekt i volymen, en exe-fil för plc-programmet etc.

Man bygger genom att aktivera 'Functions/Build Node' i konfiguratorns meny, eller motsvarande knapp i verktygspanelen. Om man har flera noder definierade i projektet måste man även välja nod i den lista som visas.

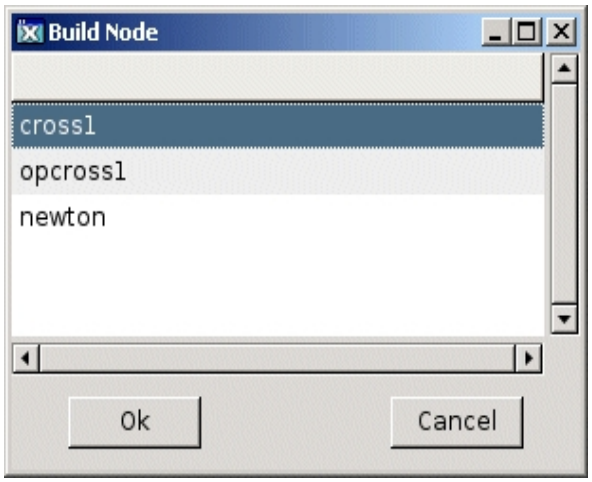


Fig Val av nod

Också bygget är uppdelat i metoder för olika klasser. Byggmetoden för ett PlcPgm är att generera kod för de fönster som är ändrade och kompilera dessa. Byggmetoden för en volym är att anropa byggmetoderna för alla objekt i volymen, samt att skapa en laddatafil för volymen. Byggmetoden för en nod är att anropa byggmetoden för volymen, samt skapa en boot-fil och länka plc-programmet. Byggmetoden för ett objekt kan anropas genom att välja ut objektet och aktivera 'Functions/Build Object' i menyn, och byggmetoden för aktuell volym genom att aktivera 'Functions/Build Volume'.

Här följer en beskrivning på några byggmetoder.

XttGraph	Kopierar pwg-filen för grafen från $pwrp_pop till $pwrp_exe.
WebGraph	Kopierar pwg-filen från $pwrp_pop till $pwrp_web.
OpPlaceWeb	Genererar html-filer för nodens hemsida och konverterar xtt-hjälp filen till web format (html).
PlcPgm	Genererar c-kod som kompileras av c-kompilatorn.
RootVolume	Anropar byggmetoden för all objekt i volymen, samt skapar en laddatafil med info om volymens objekt. Skapar även korsreferens filer om detta är specificerat i Options.
ClassVolume	Genererar include-filer med c-struct'ar för volymens klasser, samt skapar en laddatafil med typ och klassbeskrivningarna.
Node	Anropar byggmetoden för rotvolymen om volymen är tillgänglig. Skapar en bootfil med info om vilka volymer som ska laddas in och länkar plc-programmet för noden.

Normalt kontrollerar byggmetoderna först om något är ändrat, och utför endast bygget om den hittar någon förändring. I vissa fall vill man tvinga ett bygge, och sätter då 'Force' i Build kolumnen i options som öppnas från 'Options/Settings' i menyn. Här kan man även markera om man vill skapa korsreferens-filer vid bygge av en volym, eller om man vill kompilera och länka plc-programmet med debug.

Om man har använder subvolymer eller delade volymer måste dessa byggas med 'Build Volume' innan noden byggs.

19.2 Simulering

Simulering innebär att man startar upp systemet för en process eller operatörs station på utvecklingsstationen. På detta sätt kan man snabbt testa program och bilder under utvecklings-fasen. Även när ett system har tagits i drift kan man testa ändringar i systemet innan man laddar ner till produktions systemet.

Vid simulering måste indata från processen simuleras, och det gör man genom skapa ett PlcPgm som läser data som ställs ut till processen, dvs Do och Ao, och utifrån dessa sätter värden på Ai och Di objekt. Det finns speciella StoDi och StoAi objekt för detta som endast används vid simulering. Man måste se till att simulerings-programmet enbart exekverar vid simulering, t ex genom att sätta ScanOff attributet i PlcWindow objektet om IOSimulFlag i IOHandler objektet inte är 1.

För att kunna starta en simulering på utvecklingsstationen den konfigureras i directory volymen. Simuleringen sker på en separat QCom bus vilket konfiguras med ett BusConfig som läggs på topnivån i directoryvolymens högra fönster. I detta anges även en busidentitet, t ex 999. Under BusConfig objektet lägger man ett NodeConfig objekt för utvecklingsstationen, och fyller i nodnamen och ip-adress. Man kan använda loopback adressen 127.0.0.1 om man inte avser att kommunicera med andra noder. Vidare måsta man ange vilken volym man vill simulera genom att sätta volymen som namn på RootVolumeLoad objektet under NodeConfig objektet.

Notera att konfigurerings guiden för directoryvolymen normalt skapar en simuleringsbus och en simuleringsnod.

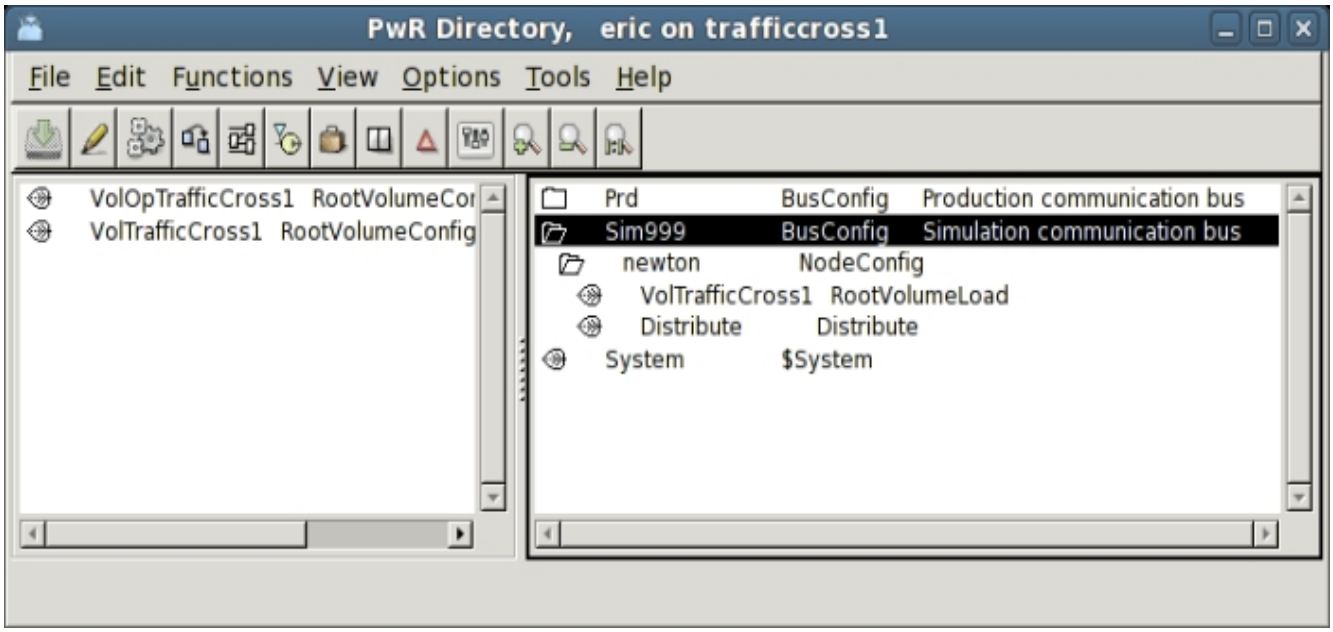


Fig Directoryvolymen med konfigurering för utvecklingsstation newton

När utvecklingsstationen är konfigurerad, bygger man genom att öppna konfiguratören för den volym som ska simuleras, aktivera 'Build Node' och välja utvecklingsstationen in i listan av noder som visas.

Innan man startar måste man definera omgivningsvariablen PWR_BUS_ID till identiteten för simuleringsbussen. Defaultvärdet för PWR_BUS_ID anges i filen /etc/proview.cnf, parameter QcomBusId. Med kommandot

```
> echo $PWR_BUS_ID
```

kontrollerar man bussidentiteten, och med kommandot

```
> export PWR_BUS_ID=999
```

sätter man ett annat värdet. Det här kommandot kan t ex läggas in i $pwrp_login/login.sh.

Nu kan man starta ProviewR runtime med

```
> rt_ini &
```

och stoppa med

```
> . pwr_stop.sh
```

Om ProviewR inte startar kan man addera -i till start-kommandot för att se eventuella felutskrifter

```
> rt_ini -i
```

Observera att man alltid måste nollställa genom att köra '. pwr_stop.sh' kommandot före ett nytt startförsök.

När runtime-mijön är startad kan man utforska systemet genom att starta runtime navigatören.

```
> rt_xtt
```

Man kan även använda runtime-monitorn för att starta runtime-miljön. Se nedan.

19.2.1 Simulerings-server

En simulerings-server kan startas för att kontrollera exekveringen av plc-programmet, och för att kunna lagra det nuvarande tillståndet av databasen och återskapade vid ett senare tillfälle.

Simulerings-servern, rt_sim, konfigureras med ett SimulateConfig objekt. I runtime hanteras servern från objektsbilden för SimulateConfig objektet.

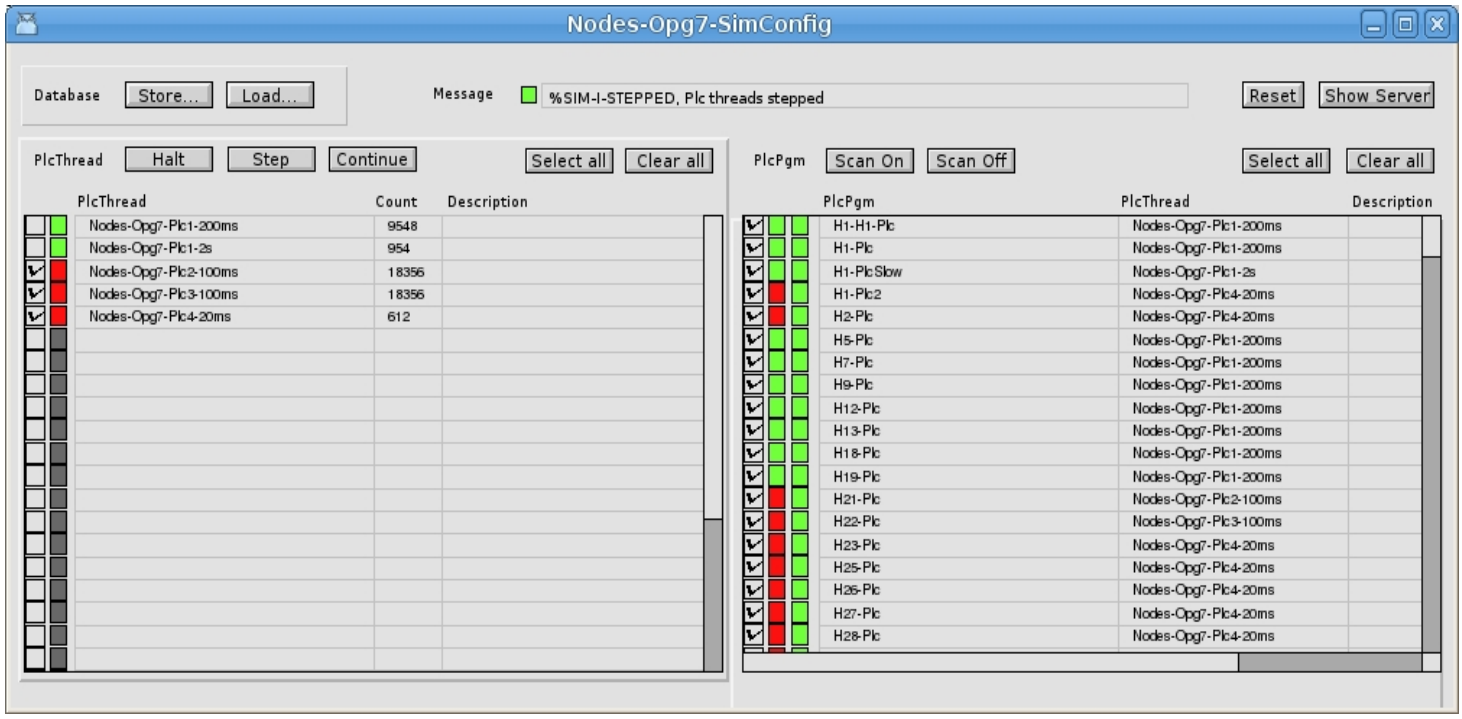
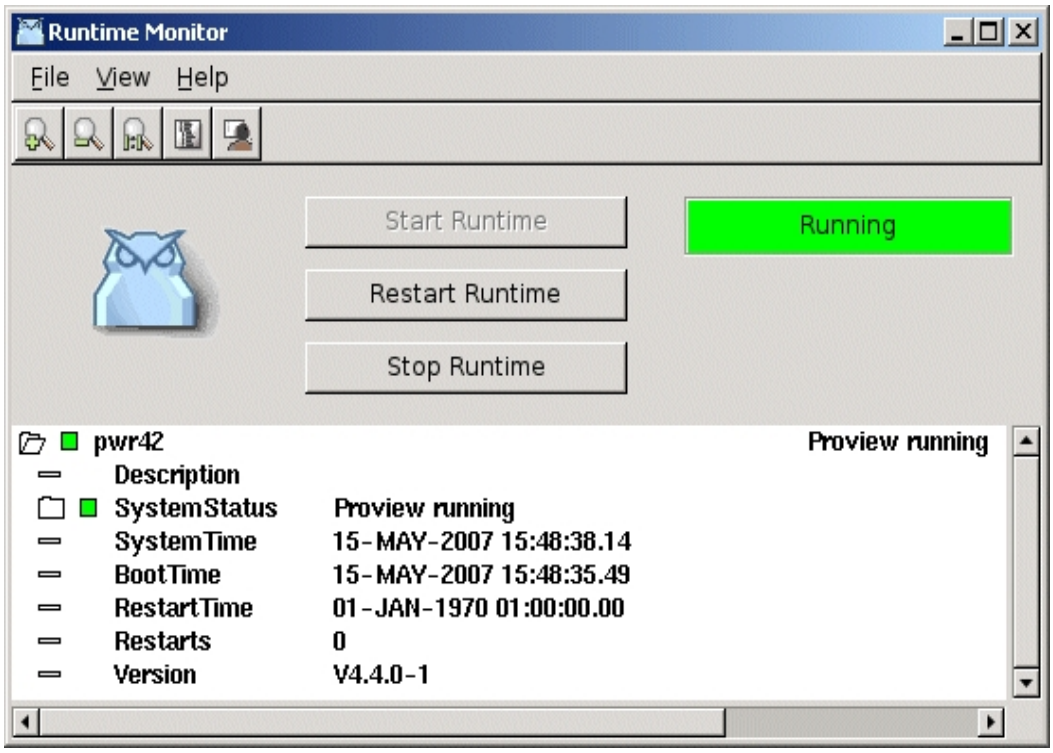


Fig Simulerings-server

19.3 Runtime Monitorn

Ofta vill man starta runtime-miljön på utvecklings-noden, t ex om man har gjort en ändring i systemet som man vill testa innan man skickar ner den till produktions-systemet.

Runtime Monitorn används för att starta och stoppa runtime-miljön på utvecklingsnoden, ofta i samband med simulering och testning, men även vid små system där utvecklingsnod och process- eller operatörs nod är en och samma dator.



Förutsättningen för att man ska kunna starta runtime-miljön är att utvecklingsnoden är

- noden är konfigurerad med ett NodeConfig objekt i projektvolymen.

- att rätt kommunikationsbuss är uppsatt. Det gör man genom att sätta omgivningsvariabeln PWR_BUS_ID till den buss man har angivit i BusConfig objektet i projektvolymen. t ex

```
export PWR_BUS_ID=999
```

För att runtime monitorn ska fungera krävs dessutom att det finns ett StatusServerConfig objekt under $Node objektet i den volym som startas.

Runtime Monitorn startas från Tools/Runtime Monitor i menyn. Här finns knappar för att starta och stoppa runtime-miljön ('Start Runtime' resp 'Stop Runtime'). I färgade rutan till höger visas om runtime är startat eller inte ('Running' resp 'Down'). Färgen indikerar status för systemet, rött för felstatus, gult för varning, och grönt för OK.

Knappen 'Restart Runtime' gör en mjuk omstart, och kan användas om runtime redan är startad. Observera att vissa ändringar inte följer med vid en mjuk omstart, t ex ändrade värden på attribut i objekt som redan finns sedan tidigare.

19.4 Process och operatörsstationer

I det här avsnittet går vi igenom det punkter som måste genomföras för att ProviewR ska kunna startas på en process eller operatörsstation.

- konfigurering av noden i directoryvolymen.
- noden ska byggas.
- installation av runtime paketet.

- distribuering till noden.
- ange bus id för noden.
- uppstart av runtime-miljön på noden.

Konfigurering in directory volymen

Noden måste vara konfigurerad i directory volymen med ett NodeConfig objekt. Detta läggs under ett BusConfig objekt som innehåller bus identiteten för produktionsbussen. I NodeConfig objektet konfigurerar man nodnamn och IP-adress, och under NodeConfig objektet ligger ett RootVolumeLoad objekt som anger den rotvolym som ska startas i noden.

Noden byggs från rotvolymens konfiguratör genom att aktivera 'Functions/Build Node' och välja ut noden i listan på noder (om det endast finns en nod konfigurerad visas inte listan).

Installation av runtimepaketet

Runtime-miljön installeras på process/operatörs-noden genom att installera pwrrt-paketet för aktuell Linux distribution. Läs installationsguiden på Download-sidan på www.proview.se för mer information.

19.4.1 Distribuera

Att distribuera innebär att man samlar ihop de filer som skapas vid bygget av en nod, och som behövas för att kunna köra runtime-miljön, till ett paket. Paketet kopieras till process eller operatörsstationen och packas upp där.

Vilka filer som ingår i paketet konfigueras i directoryvolymen med ett Distribute objekt som läggs under NodeConfig objektet för noden. Distribute-objektet innehåller attributet Components i där man anger vilken typ av komponenter eller filer som ska ingå. Om komponenter är angivna som inte finns genererade resulterar det i varningsutskrifter vid distributionen.

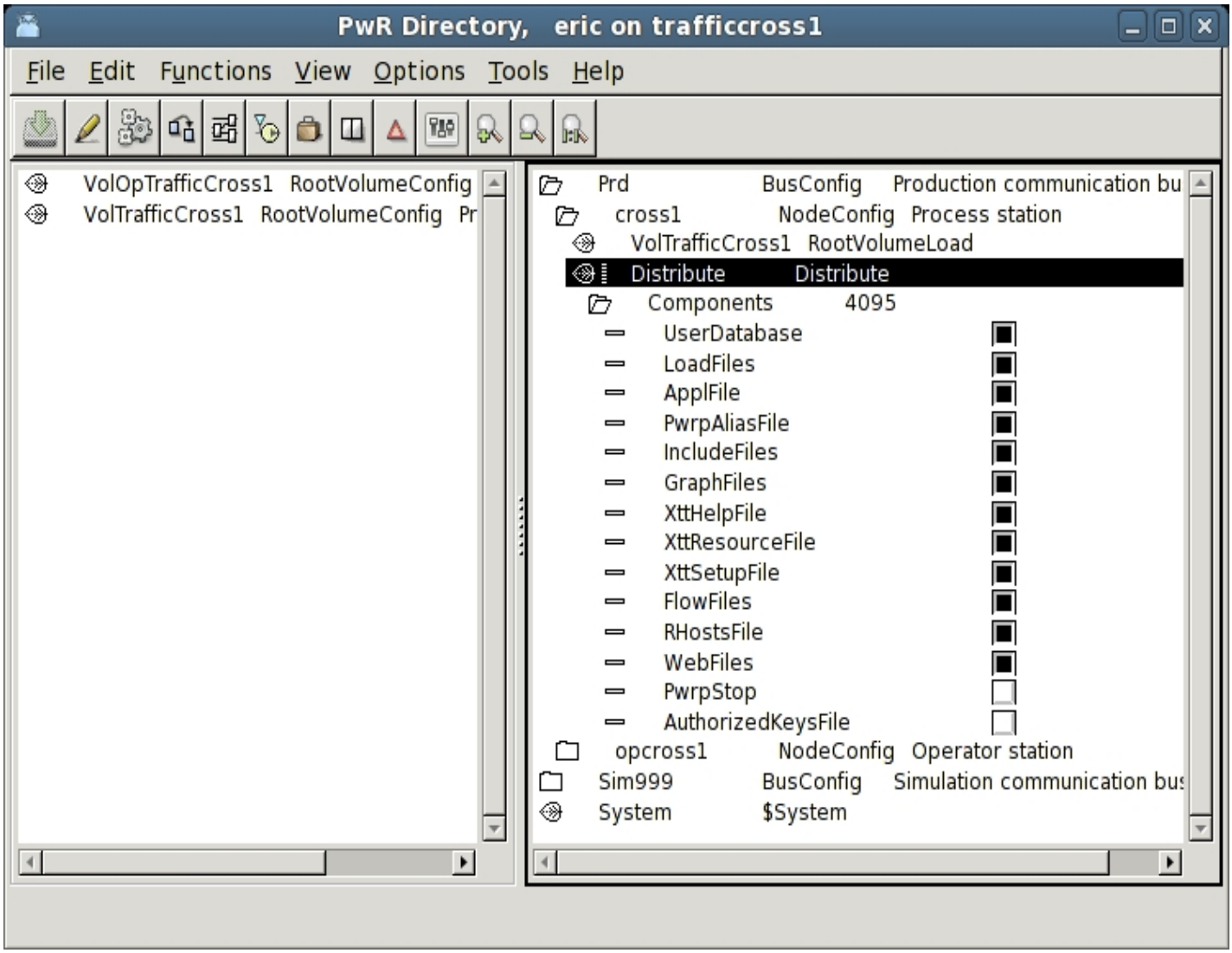


Fig Distribute objektet

Om man har andra filer, t ex applikations-program, som ska ingå i paketet, lägger man ett objekt av typen ApplDistribute under Distribute objektet. I ApplDistribute objektet kan man ange den eller de filer (specification med wildcard är tillåten) som ska adderas, samt till vilken katalog de ska kopieras.

Alla filer som behövs i runtime bör ingå i paketet. Dels är det viktigt att ett paket representerar en fullständig version av systemet, så att runtimemiljön kan återskapas, t ex om man vill backa tillbaka till en tidigare version. Råkar man ut för en disk-krash är det också viktigt att allt finns i paketet, och att det inte behövs diverse handpåläggning för att systemet ska starta med en ny disk.

Distributionen utförs från Distributören som öppnas från 'Functions/Distribute' i konfiguratorns meny. Välj ut den nod som ska distribueras till och aktivera 'Functions/Distribute' distributörens menyn.

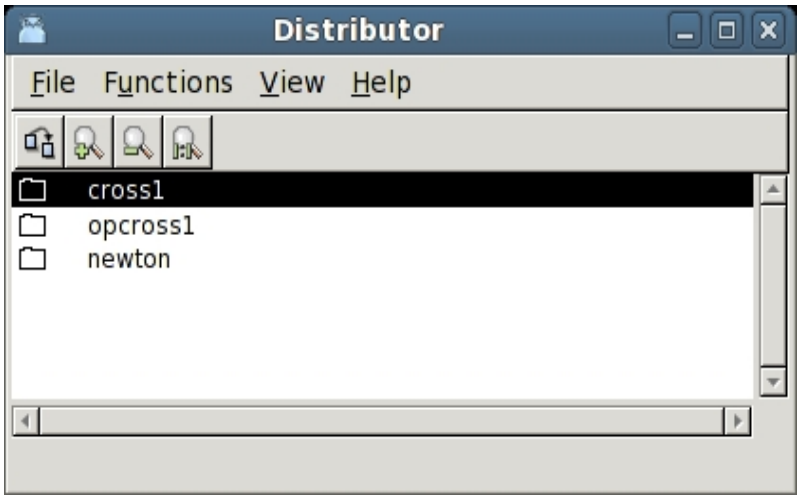


Fig Distributören

Distributören samlar nu ihop angivna filer, skapar ett paket och kopierar till användaren 'pwrp' på processnoden. ssh kräver att man anger passerord, detta anges två gånger i det terminalfönster från vilken konfiguratören är startad. Vid installationen har användaren 'pwrp' passerordet 'pwrp' men detta kan ha ändrats av säkerhetsskäl.

Om man inte har nätverkskontakt med processnoden, kan man skapa ett paket och flytta det t ex med en USB sticka. Välj ut noden i distributören, och aktivera 'Functions/Create Package'. Paketet lagras på $pwrp_load med namnet pwrp_pkg_'nodename'_'version'.tgz, t ex pwrp_pkg_cross1_0002.tgz. På processtationen packar man upp med skriptet pwr_pkg.sh -i och anger namnet på paketet.

```
> pwr_pkg.sh -i pwrp_pkg_cross1_0002.tgz
```

Backa till tidigare version

Ibland fungerar inte en ändring som planerat och man vill backa tillbaka till föregående version. Detta gör man enkelt med pwr_pkg.sh. Paketen lagras på användaren pwrp hemma-katalog /home/pwrp. Genom att leta upp det paket som tidigare var installerat och starta pwr_pkg.sh -i med detta återställer man till denna version.

```
> pwr_pkg.sh -i pwrp_pkg_cross1_0001.tgz
```

19.4.2 Bus identitet

QCom bus identitet för noden är angiven i filen /etc/proview.cnf.

```
# Default QCOM Bus Id
qcomBusId 517
```

Det här värdet ska överensstämma med den konfigurerade busidentiteten i directory volymen. Den aktuella busidentiteten är lagrad i omgivningsvariabeln $PWR_BUS_ID.

19.4.3 Starta runtime-miljön

Runtime-mijön startas med kommandot

```
> pwr start
```

och stoppas med

```
> pwr stop
```

eller

```
> pwr kill
```

Kommandot 'pwr stop' kräver att alla processer lever, medan 'pwr kill' är lite tuffare och rensar i alla lägen.

Om ProviewR runtime inte startar, kan man starta med kommandot

```
> rt_ini -i
```

för att visa konsol-loggning i terminalfönstret.
Återställ alltid med 'pwr kill' före ett nytt startförsök.

20 Konfiguratören

Konfiguratören används för att navigera och konfigurera i arbetsbänken.
Konfiguratören visar objekten i en volym. Objekten delas vanligen upp i två fönster, ett till vänster och ett till höger, och hur uppdelningen görs beror på vilken typ av volym som hanteras.

- För rotvolymer och subvolymer visas anläggningshierakin i det vänstra fönstret och nodhierkin i det högra. Närmare bestämt visas toppobjekt av klassen $PlantHier i det vänstra och toppobjekt av klassen $NodeHier i det högra.

- För directory volymen visas volymer i det vänstra fönstret och bussar och noder i det högra. Även systemobjektet visas i det undre fönstret.

- För klassvolymer visas klasser i det vänstra fönstret och typer i det högra.

Med 'View/TwoWindow' kan man välja om man vill visa två fönster eller bara ett. Om endast ett fönster visas kommer det varannan gång man aktiverar 'TwoWindows' att vara det vänstra fönstret som visas och varannan gång det högra.

Med 'Edit/Edit mode' går man in i editerings mode, och palett med olika klasser dyker upp till vänster. Man kan nu skapa nya objekt, flytta objekt, ändra värden på attribut mm.

Representation av volymer

Volymer kan lagras i olika format, i databas, i laddatafil eller i en textfil. Konfiguratören kan visa en volym i alla dessa format och kan ha fyra olika representationer av volymer:

- db, en databas. Rotvolymer och subvolymer skapas och editeras i en databas. Innan man kan starta runtimemiljön genererar man laddatafiler för volymerna. Laddatafilerne läses in vid uppstart av runtimemiljön. db-representationen är editeringsbar.

- wbl, en textfil med filtyp .wb_load. Klassvolymer lagras som wbl, och rot- och subvolymer kan dumpas på wbl-fil t ex vid uppgradering, för att sedan återladdas. wbl-representationen är inte editeringsbar. Vid editering av klassvolymer överför man wbl-representationen till en mem-representation, se nedan, för att därefter spara den som wbl igen.

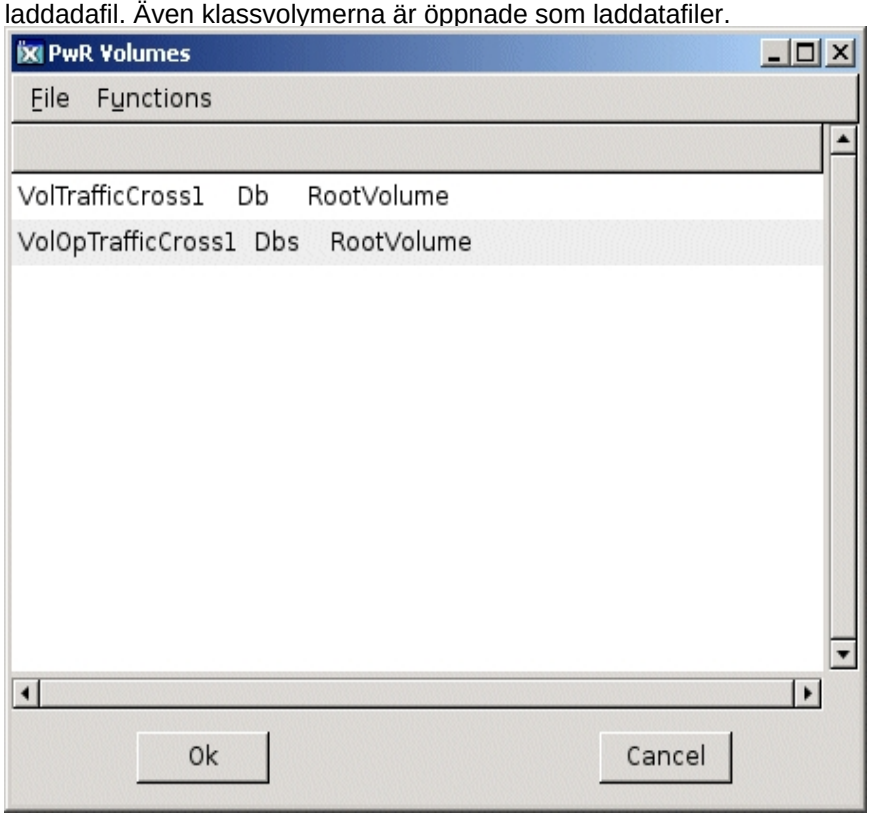
- dbs, en laddatafil. Från rotvolymer, subvolymer och klassvolymer i db resp. wbl representation skapas laddatafiler som används av runtimemiljön. Konfiguratören läser in klassvolymernas dbs-filer för att tolka klasser, och rot och subvolymers dbs-filer för att visa monterade volymer och kunna översätta referenser till externa objekt. dbs-representationen är inte editeringsbar.

- mem, en volym som konfiguratören håller internt i minnet. Copy/Paste buffrar består t ex av mem-volymer. Klasseditorn importerar klassvolymen, som ursprungligen finns som wbl, till en mem-volym, eftersom mem-representatinen är editeringsbar.

Som vi ser ovan kan samma volym finnas både som databas och som laddatafil. När man startar konfiguratören anger man en volym som argument. För denna volymen öppnas databasen, den alltså representerad som en db, för övriga volymer i projektet öppnas laddatafilerna, dessa finns representerade som dbs. Det här gör att man kan titta på övriga volymer i projektet, och man kan lösa upp referenser till dem, men det är inte editeringsbara. Är databasen för

volymen låst, pga att någon annan har öppnat den, ges ett felmeddelande och laddatafilerna öppnas istället för databasen.

I figuren nedan visas volymslistan, som öppnas från 'File/Open' i menyn. Det visar alla volymer som är öppnade i konfiguratören. Vi kan se att databasen för rotvolymen VolTrafficCross1 är öppnad, medan den andra rotvolymen, VolOpTrafficCross1 är öppnad som laddadafil. Även klassvolymerna är öppnade som laddatafiler.



Om ingen volym anges som argument när man startar konfiguratören, öppnas direktory volymens databas, och övriga volymer som dbs-volymer.

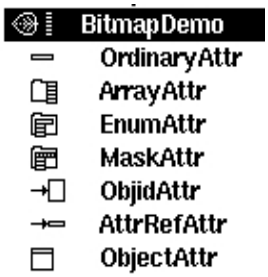
Navigera

I konfiguratören visas objekten i den aktuella volymen. Objekten ligger i en trädstruktur, och objekt som har barn visas med en mapp, objekt utan barn med ett löv. För varje objekt visas som default, dessutom objektets namn, klass och eventuell beskrivning (beskrivningen hämtas från ett Description attribut i objektet).

Genom att klicka MB1 på en mapp, öppnas mappen och objektets barn visas. Är mappen redan öppen stängs den. Man kan även öppna en map genom att dubbelklicka var som helst på raden för objektet.

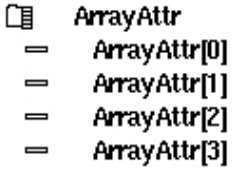
Vill man se innehållet i ett objekt klicka man med Shift/Klick MB1 på mappen eller lövet, eller Shift/Dubbelklick MB1 var som helst på objektsraden. Nu visas objektets attribut, och värdet på respektive attribut. Attributen markeras med olika figurer beroende på typ.

Bitmappar för olika typer av attribut



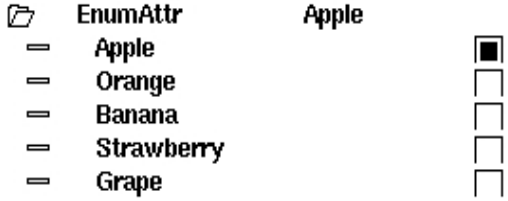
- Ett vanligt attribut markeras med en långsmal rektangel.

- En vektor markeras med en map och en stapel av attribut markeringar. Vektorn kan öppnas genom att man klickar med MB1 på mappen eller dubbelklickar någonstans på attributraden. Nu visas de olika elementen i vektorn.

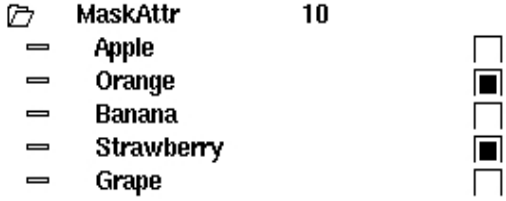


- Ett attribut som refererar till ett annat attribut eller objekt, dvs av typen Objid eller AttrRef, markeras med en pil som pekar på en fyrkant.

- Uppräkningstyper, Enum, markeras med en map med div långsmala rektanglar i. Genom att klicka MB1 på mappen visas de lika alternativen i uppräkningstypen. Alternativen visas med checkboxar med det valda alternativet markerat. Man kan även dubbelklicka med MB1 på attributraden för att se alternativen.



- Masktyper, Mask, markeras på liknande sätt som Enum, och de olika bitarna visas med klick MB1 på mappen, eller dubbelklick MB1 på attributraden.



- Attributobjekt, dvs attribut som innehåller datastrukturen för ett objekt, markeras med en fyrkant med dubbellinje på ovansidan. Attributobjektet öppnas med klick MB1 på fyrkanten, eller dubbelklick MB1 på attribut raden.

Ett objekt eller attribut väljs ut genom att man klickar med MB1 på det (dock inte på mappen/lövet). Med Shift MB1 kan man välja ut fler objekt. Genom att dra med MB1 kan man också välja ut flera objekt.

Ur ergonomisk synvinkel är det ofta bättre att navigera från tangenbordet. Man använder då främst piltangenterna. Först måste man se till att fönstret har inmatings fokus genom att klicka på det. Inmatnings fokus mellan vänster och höger fönster skiftas med TAB.

Men PilUpp/PilNer väljer man ut ett objekt. Har objektet barn öppnar man barnaskaran med PilHöger, och stänger med PilVänster. Innehållet i objektet, dvs objektets attribut, visas med Shift/PilHöger och stängs med PilVänster.

Attribut som är vektorer, enum, mask eller attributobjekt, öppnas med PilHöger och stängs med PilVänster.

När man känner sig hemma i objektsträdet kan man sätta upp sig som 'advanced user'. Ytterligare funktioner läggs då på piltangenterna. PilHöger på ett objekt visar t ex attributen i objektet om det inte har några barn. Har det barn får man använda Shift/PilHöger som förut.

Editera

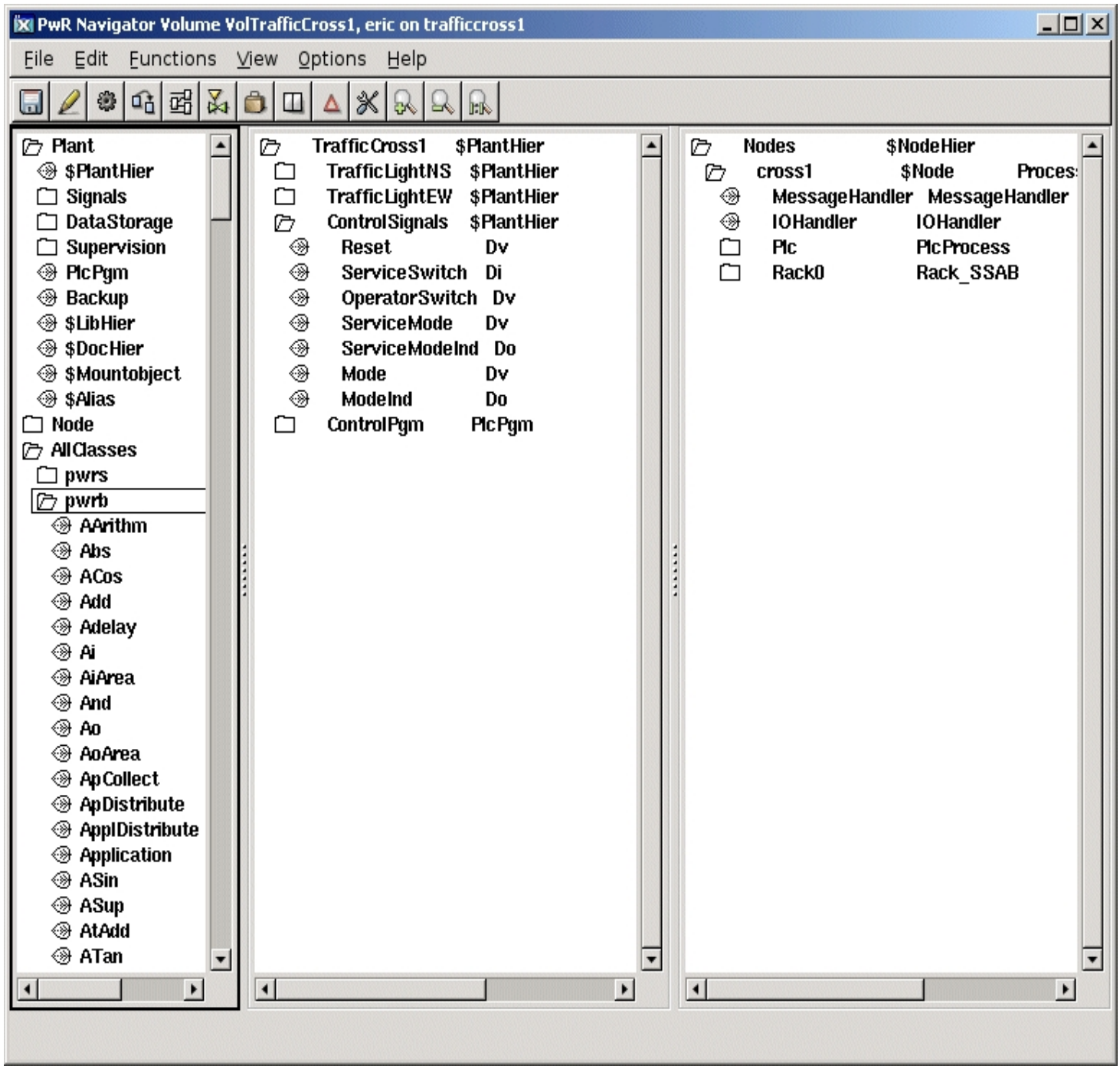
Editering innebär att man skapar nya objekt, kopierar objekt, tar bort objekt eller ändrar värde på attribut.

Skapa objekt

Man skapar ett objekt genom att välja ut objekts klass i paletten. Paletten är uppdelad i mapparna Plant, Node och AllClasses. Under Plant finns de vanligaste klasserna i anläggnings hierakin, och under Node finns de vanligaste i nodhierakin. Om klassen inte finns med här, finns samtliga klasser åtkomliga under AllClasses. Här listas alla klassvolymer, och under respektive volym, klasserna i volymen. Därefter klickar man med mittenknappen på blivande syskon eller förälder till det nya objektet. Om man klickar på mappen/lövet på destinations-objektet läggs det nya objeket som första barn, om kan klicka till höger om mappen/lövet läggs det som syskon.

Man kan även skapa objekt från popupmenyn. Välj ut en klass i paletten och öppna popupmenyn genom att högerklicka på destinationsobjektet. Aktivera 'Create Object' och välj hur det nya objektet ska läggas i relation till destinationsobjektet, före eller efter, eller som första eller sista barn.

Konfiguratören i editeringsläge



Ta bort objekt

Objekt tas bort mha popmenyn. Klicka med högerknappen på objektet och aktivera 'Delete Object'.

Flytta objekt

Man kan även flytta objekt mha popupmenyn, men det är ofta enklare att använda musens mittenknappen: välj ut det objekt som ska flyttas och klicka med mittenknappen på destinationsobjektet. Om man klickar på mappen/lövet på destinationsobjektet läggs objektet som första barn, annars som syskon.

Obs! Undvik att använda Cut/Paste för att flytta ett objekt, eftersom detta skapar en

kopia av objektet med ny objektsidentitet, och referenser till objektet kan förloras.
Man kan använda kommandot paste/keepoid för att behålla identiteten.

Kopiera objekt

Kopiering kan ske med copy/paste eller mha popupmenu.

- copy/paste. Välj ut det eller de objekt som ska kopieras och aktivera 'Edit/Copy' (ctrl/C) i menyn. De utvalda objekten kopieras nu till en paste-buffert. Välj ut ett destinations objekt, och aktivera 'Edit/Paste' (ctrl/V). Objekten i paste-bufferten läggs nu som syskon till destinationsobjektet. Om man istället aktiverar 'Edit/Paste Into' (shift+ctrl/V) läggs de nya objekten som barn till destinationsobjeketet. Om det kopierade objektet har barn, kopieras även barnen av copy/paste.

- från popupmenyn. Välj ut det eller de objekt som ska kopieras, ta upp popmenyn från destinationsobjektet, och aktivera 'Copy selected object(s)'. Man har här möjlighet att välja hur de nya objekten ska läggas relativt destinationsobjektet, som första eller sista barn, eller som nästa eller föregående syskon. Om de koperade objekten har ättlingar, och även de ska kopiera aktiverar man 'Copy selected Tree(s)' istället.

Ändra namn på ett objekt

Namnet ändras genom att objektet väljs ut och 'Edit/Rename' (ctrl/N) aktiveras i menyn. Ett inmatningsfält öppnas längs ner i konfiguratören där det nya namnet matas in. Ett objektsnamn kan ha max 31 tecken.

Man kan även ändra namn genom att visa objektets attribut. När man är i editeringsläge visas nu objektsnamnet främst bland attributen, och kan ändras pss som ett attribut.

Ändra värde på ett attribut

Välj ut det attribut som ska ändras, och aktivera 'Functions/Change value' (ctrl/Q) i menyn. Ange de nya värdet i inmatningsfältet. Vill man avbryta inmatningen aktiverar man 'Change value' igen.

Alla attribut är inte ändringsbara. Det beror på attributets funktion, om det ska datasättas i utvecklingsmiljön, eller inte. De som är ändringsbara markeras med en pil.

Man kan även ändra värde på attribut från objektseditorn som öppnas från popupmenyn (Open Object). Attribut som är av typen flerradig text, kan endast ändras från objekteditorn.

Som 'advanced user' kan man öppna inmatningsfältet med PilHöger, som ett snabbare alternativ till 'Change value'.

Symbol fil

Symbol filen är en kommando fil som exekveras vid startup av konfiguratören. Det kan innehålla definitioner av symboler och andra wtt kommandon. Här följer några exempel på användbara kommandon.

Genväg till objekt i objektsträdet:

```
define rb9 "show children /name=hql-rb9"
```

20.1 Objektseditorn

Datamängden för ett objekt består av attribut. Objektseditorn visar attributen i ett objekt och värdet på varje attribut. Om man är i editeringsmod kan man även sätta värden på attributen.

Attributen visas på samma sätt som i konfiguratören, skillnaden är att de visas i ett eget fönster.

Navigera

Navigering och datasättning sker också pss som i konfiguratören.

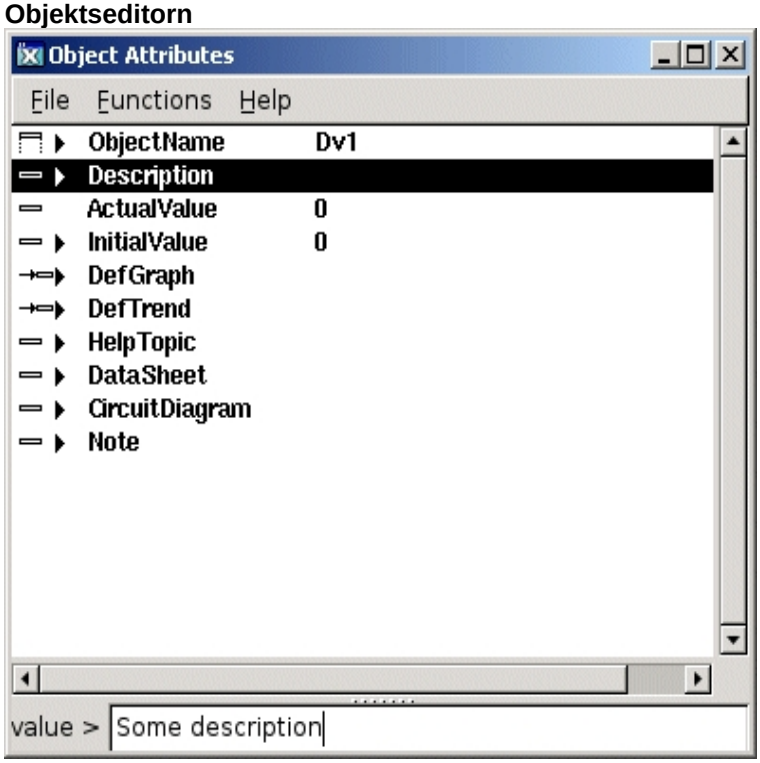
Start

Objektseditorn startas från konfiguratören eller plceditorn. Aktivera OpenObject i popupmenyn för ett objekt, eller välj ut objektet och aktivera 'Functions/Open Object' i menyn. I Plceditorn kan man även starta objektseditorn genom att dubbelklicka på objektet. Om konfiguratören/plceditorn är i editeringsmod öppnas även objektseditorn i editeringsmod.

Meny

File/Close	stäng objektseditorn.
Functions/Change value	öppna inmatningsfältet för det utvalda attributet. Detta kan endast göras i edit mode.
Functions/Close change value	stäng inmatningsfältet.

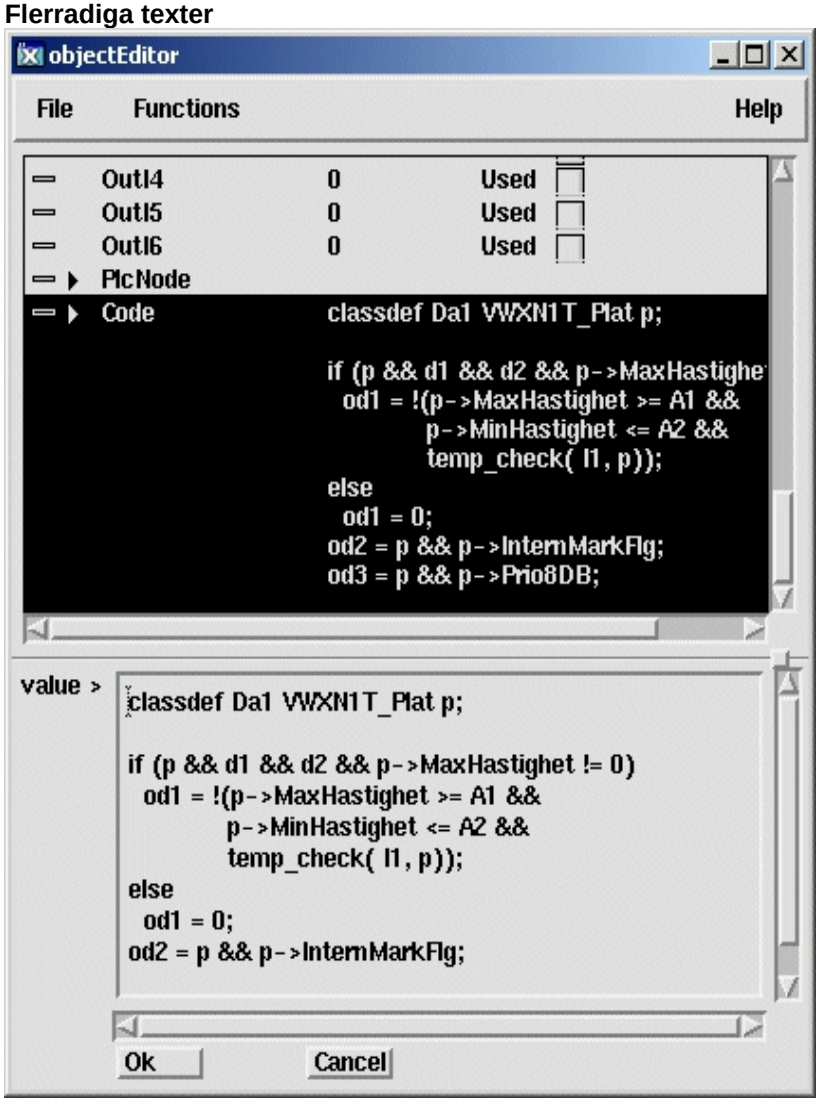
Objektseditorn



Objektseditorn har inmatingsfält för att mata in flerradiga texter. I det här fallet kan man inte avsluta inmatingen med 'Enter' som med enradiga texter. Antingen klickar man på 'Ok'

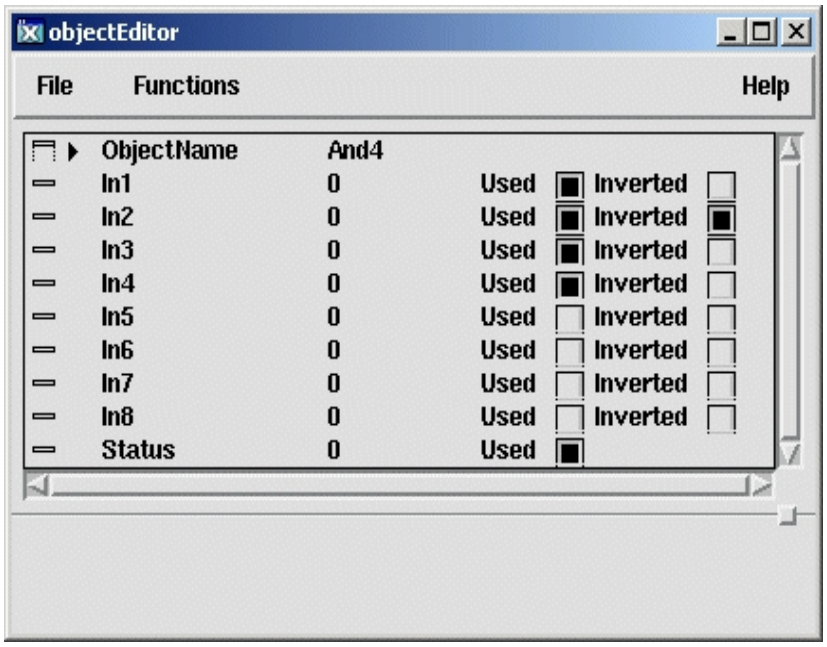
knappen, eller också aktiverar man 'Functions/Close change value' (Ctrl/T) för att avsluta.

Flerradiga texter



Objektseditorn för plc-objekt har funktioner för att ange vilka in resp utgångar som ska visas i funktionsobjektet. Man kan även välja om digitala ingångar ska vara inverterade. Detta väljs med checkboxar för respektive attribut ('Used' resp 'Inverted'). Checkboxen för 'Used' kan även ändras från tangentbordet med 'Shift/PilHöger' och checkboxen för 'Inverted' kan ändras med 'Shift/PilVänster'.

Plc objekt med checkboxar



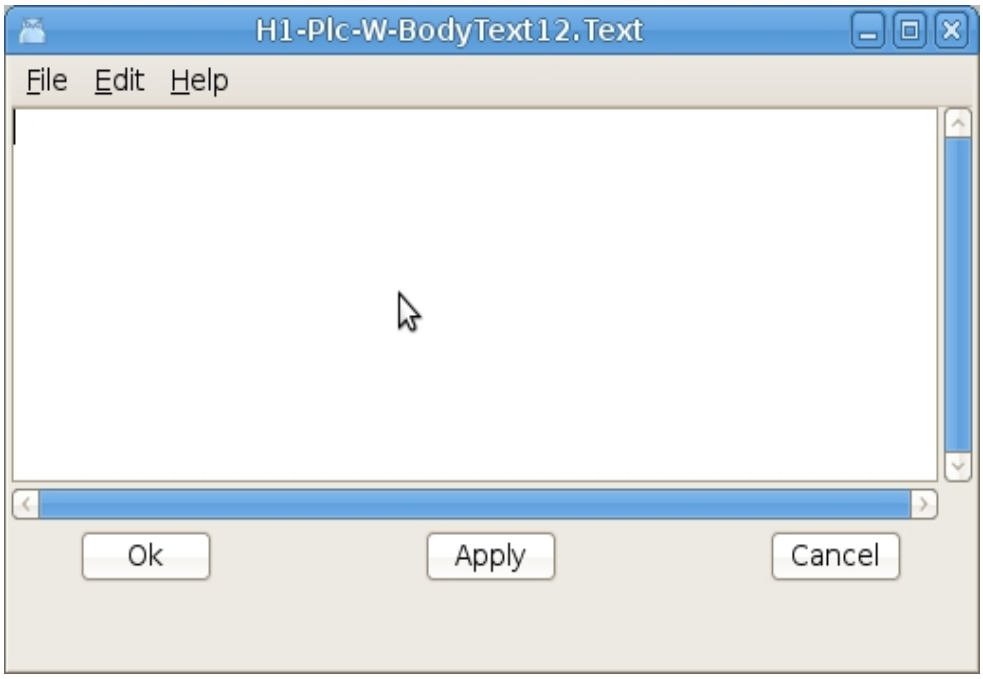
20.2 Texteditor

För text-attribut i objekt som BodyText, HelpText, CArithm och DataArithm finns en speciell texteditor. Editorn öppnas genom att högerklicka på objektet i plc-editorn och aktivera EditText eller EditCode i popupmenyn.

Meny

File/Close	Stäng texteditorn.
File/Save	Spara texten i objektets attribut. Notera att texten är inte sparad i databasen förrän sessionen i plceditorn har sparats.
Edit/Copy	Kopiera utvald text till urklipp.
Edit/Cut	Klipp ut utvald text.
Edit/Paste	Klistra int text från urklipp till aktuell markör-position.

Texteditor



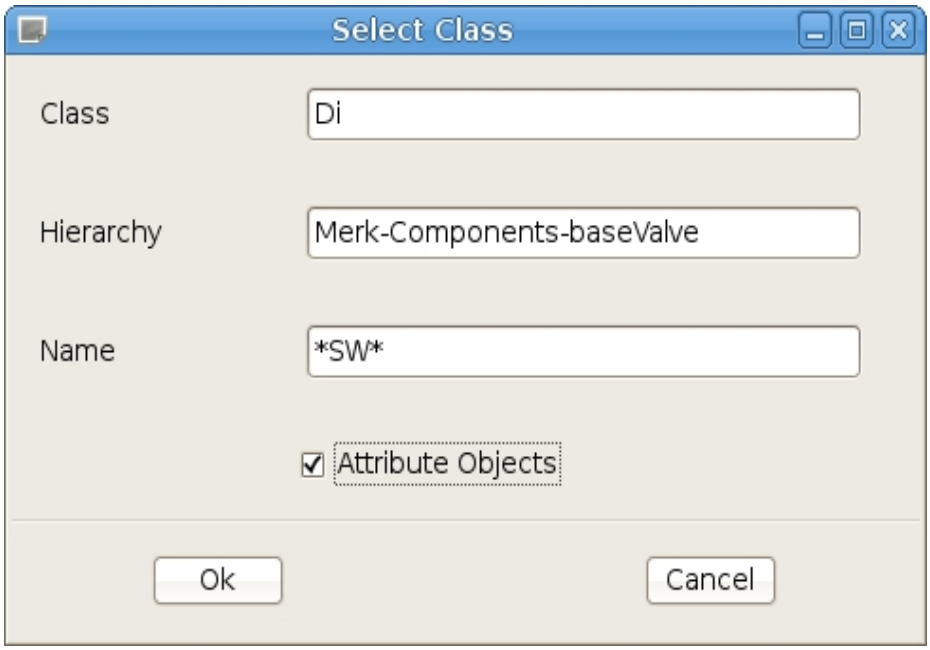
20.3 Spreadsheet editorn

Spreadsheet editorn används för att titta på, eller konfigurera, flera objekt av samma klass samtidigt. Objekt av en viss klass, under en specifierat objekt i objektsträdet, visas i en tabell i editorn. I tabellen visas även värdet på ett attribut i objekten, och man kan enkelt skifta mellan olika attribut.

Spreadsheet editorn startas från konfiguratören: 'Functions/Spreadsheet' i menyn.
Om konfiguratören är i editeringsmod, startas även spreadsheet editorn in editeringsmod.

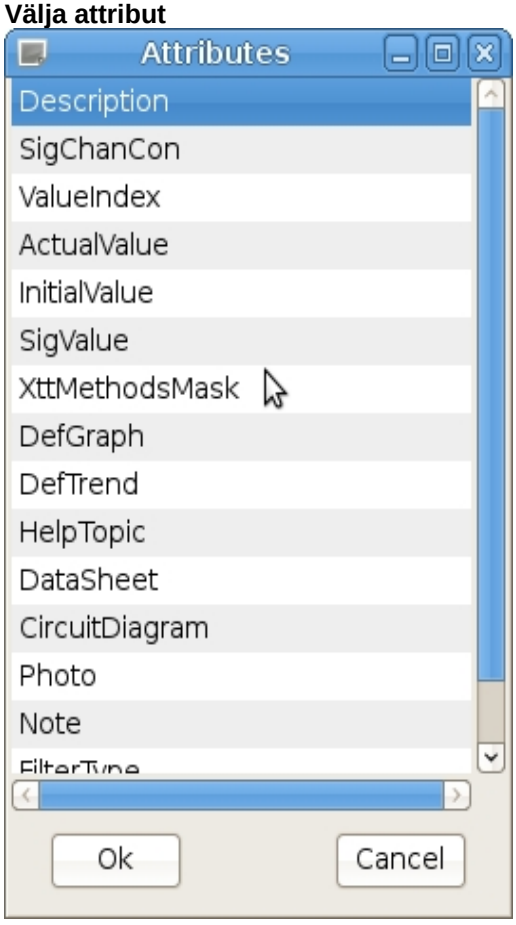
När spreadsheet editorn är startad, måste man först ange vilka objekt som ska visas: vilken klass de tillhör och under vilken hierarki de ska ligga. Det här görs genom att aktivera 'File/Select Class' i menyn. Mata in klass, hierarki, samt ange om attributobjekt, dvs objekt som ligger som attribut i andra objekt, ska vara med. Man kan även välja objekt efter namn med en wildcard-sträng.

Välja klass och hierarki



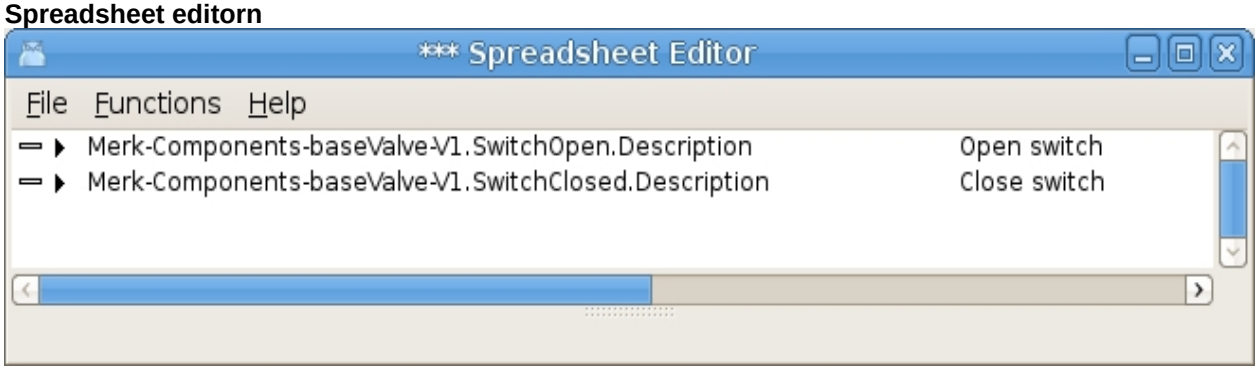
Därefter väljer man vilket attribut som ska visas. Välj ut ett attribut i attributlistan och klicka på 'Ok', eller dubbelklicka på ett attribut.

Välja attribut



Resultatet visas i figuren nedan. Här valdes attributet 'Description'. Man kan enkelt se övriga attribut i objekten genom att aktivera 'File/Next Attribute' (Ctrl/N) och 'File/Previous Attribute' i menyn.

Spreadsheet editorn



Meny

File/Select Class	ange klass och hierarki för de objekt som ska visas.
File/Select Attribute	ange vilket attribut som ska visas.
File/Next Attribute	visar nästa attribut för objekt i tabellen.
File/Previous Attribut	visar föregående attribut för objekt i tabellen.
File/Print	skriver ut tabellen.
File/Close	stänger spreadsheet editorn.
Functions/Change value	öppna inmatningsfält för utvalt objekt.
Functions/Close change value	stäng inmatningsfältet.

20.4 Hjälp fönster

Hjälpfönstret används för att visa och navigera i hjälptexter. Hjälptexterna kan vara olika typer av manualer och handböcker som följer med ProviewR, eller hjälptexter som skrivs av konstruktören för att beskriva anläggningen och ge operatörerna assistans.

20.5 Meddelande fönster

Meddelandefönstret visar meddelanden från ProviewR som ges vid olika operationer. Meddelandena kan ha fem olika nivåer av allvarlighet, som markeras med olika färger:

S	Framgång (Success)	grön
I	Information	grön
W	Varning (Warning)	gul
E	Fel (Error)	röd
F	Fatalt fel (Fatal)	röd

Om en pil visas före meddelandet innehåller meddelandet en länk till ett objekt. Genom att klicka på pilen letas det aktuella objektet upp.

20.6 Utilities fönstret

Utilities fönstret är ett grafiskt gränssnitt till olika kommandon i konfiguratören.
För en närmare beskrivning av respektive kommando, se avsnittet Kommandon.

20.7 Backup verktyg

Backupverktyget analyserar en runtime backupfil. Den gör det möjligt att

- inspektera innehållet i en backupfil.
- se skillnaden mellan två backupfiler lagrade vid olika tidpunkt.
- se skillnaden mellan en backupfil och motsvarande värden i utvecklingsdatabasen.
- överföra utvalda värden från backupfilen till utvecklingsdatabasen.

Backupfunktionen i ProviewR runtime, lagrar värden på objekt och attribute specificerade med Backupobjekt, till backupfilen. Backupfilen läses vid ProviewR uppstart och de tidigare lagrade värdena läggs in i realtidsdatabasen.

Visa innehållet i en backupfil.

Öppna backupfönstret från konfiguratorn för rotvolymen för den node från vilken backupfilen har hämtats. Backupfönstret öppnas med kommandot 'backup show'. Aktivera File/Open i menyn och välj en backupfil. Backupfilen kan vara en kopia av den nuvarande backupfilen eller en kopia från ett tidigare tillfälle. Alla attribut och deras värden visas i backupfönstret.

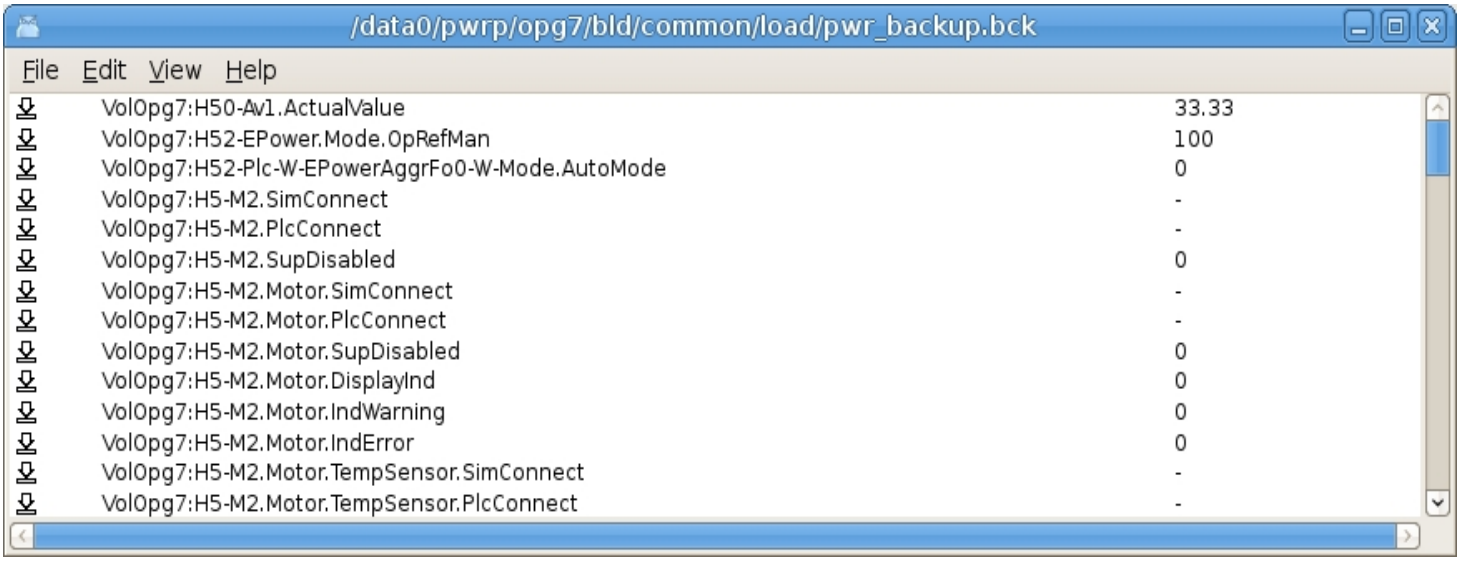


Fig Visa backupfilen

Jämför två backupfiler

Öppna den första backupfilen för visning som beskrivs ovan. Aktivera sedan File/Compare Backup File i menyn och välj den andra backupfilen. Attribut med värden som skiljer sig mellan filerna visas nu.

Jämför backupfilen med utvecklingsdatabasen

Öppna backupfilen för visning, och aktivera sedan File/Compare Databse in menyn. Attribut med värden som skiljer sig mellan backupfilen och utvecklingsdatabasen visas nu.

Överför värden från en backupfil till utvecklingsdatabasen

Gå till editeringsmod i konfiguratorn, och öppna sedan backupfönstret. Öppna

backupfilen för visning, och jämför den med värden från utvecklingsdatabasen med File/Compare Database in menyn. Attribut med värden som skiljer mellan backupfilen och utvecklingsdatabasen visas nu med checkboxar. Markera de värden som ska överföras till databasen, och activera File/Transfer to database i menyn. Aktivera även Save i konfiguratorn för att spara ändringarna.

20.8 Build Directories

Build Directories fönstret visar kataloger som är konfiguerarde att byggas genom att kopiera filer, exekvera makefiler, skript och program. Kataloger som innehåller element som behöver uppdateras är markerade i fönstret.

OBS! Om en fil lagras efter Build Directories fönstret öppnades, måste uppdaterings-knappen aktiveras för att visa korrect status. Annars kommer den lagrade filen inte att uppdateras när katalogen byggs.

Mappen för en katalog kan öppnas för att visa filer som ska uppdateras. Genom att aktivera 'Edit/Show all' i menyn visas även filer som inte kommera att updateras.

Bygget exekveras endast för kataloger och filer som är markerade. Enstaka kataloger eller filer kan byggas genom att avmarkera övriga kataloger och filer.

Genom att trycka på 'Build Directories' i verktygspanelen utförs bygget.

Katalogerna och bygg-åtgärderna konfigureras i directory volymen.

20.9 Build Export och Import

Build Export fönstret visar filer som ska exporteras till andra projekt och moduler. Exempel på gemensamma filer för projekt är dbs-filer för klassvolymer och h-filer för transaktioner. Vanligvis exporteras filer till en gemensam katalog, varifrån de sedan importeras av andra project. Det är inte rekommendabelt att importera eller exportera direkt till eller från andra projekt.

Genom att trycka på 'Export files' knappen i verktygspanelen utförs exporteringen.

Build Import fönstret visar filer som ska importeras från andra projekt och moduler.

Genom att trycka på 'Import files' knappen i verktygspanelen utförs importeringen.

21 Plceditorn

Plceditorn används för att skapa plcprogram mha ett grafiskt programmeringsspråk.

Programmeringen sker med funktionsblock som läggs i ett horisontellt nät av noder och förbindningar från vänster till höger. Signaler eller attribut hämtas upp på den vänstra sidan av nätet, och värdena skickas vidare med förbindningar från utgångs-pinnar till ingångs-pinnar på funktionsblocken. Funktionsblocken opererar på värdena och på nätets högra sida, lagras värdena i signaler eller attribut.

Grafcet sekvenser består av ett vertikalt nät av noder och förbindningar. Ett tillstånd förflyttas mellan olika steg i sekvensen via förbindningarna. Grafcet och funktionsblock kan samverka med varandra och kombineras till ett enda nät.

Start

Plceditorn startas från konfiguratören. Välj ut ett objekt av klassen PlcPgm och aktivera 'Functions/Open Program' (Ctrl/L) i menyn, eller aktivera 'Open Program' i popupmenyn för PlcPgm objektet. Konfiguratören ska inte vara i editeringsmod.

Arbetsmod

Plceditorn kan vara i fyra olika moder: View, Edit, Trace och Simulate. Moden väljs under 'Mode' i menyn.

View
View innebär att plceditorn är i visningsmod. Man kan inte skapa eller modifiera objekt. Många av menyalternativen som har med editering att göra är dimmade.

Edit
Om man har editeringsprivileger kan man gå över i editeringsmod. Här kan man skapa och modifiera objekt.

Trace och Simulate
Om man vill felsöka programmet går man in i tracemode. Detta kräver att proview's runtime miljö är startad på utvecklingsstationen. Simulate fungerar som Trace, man man kan dessutom sätta värden på signaler.

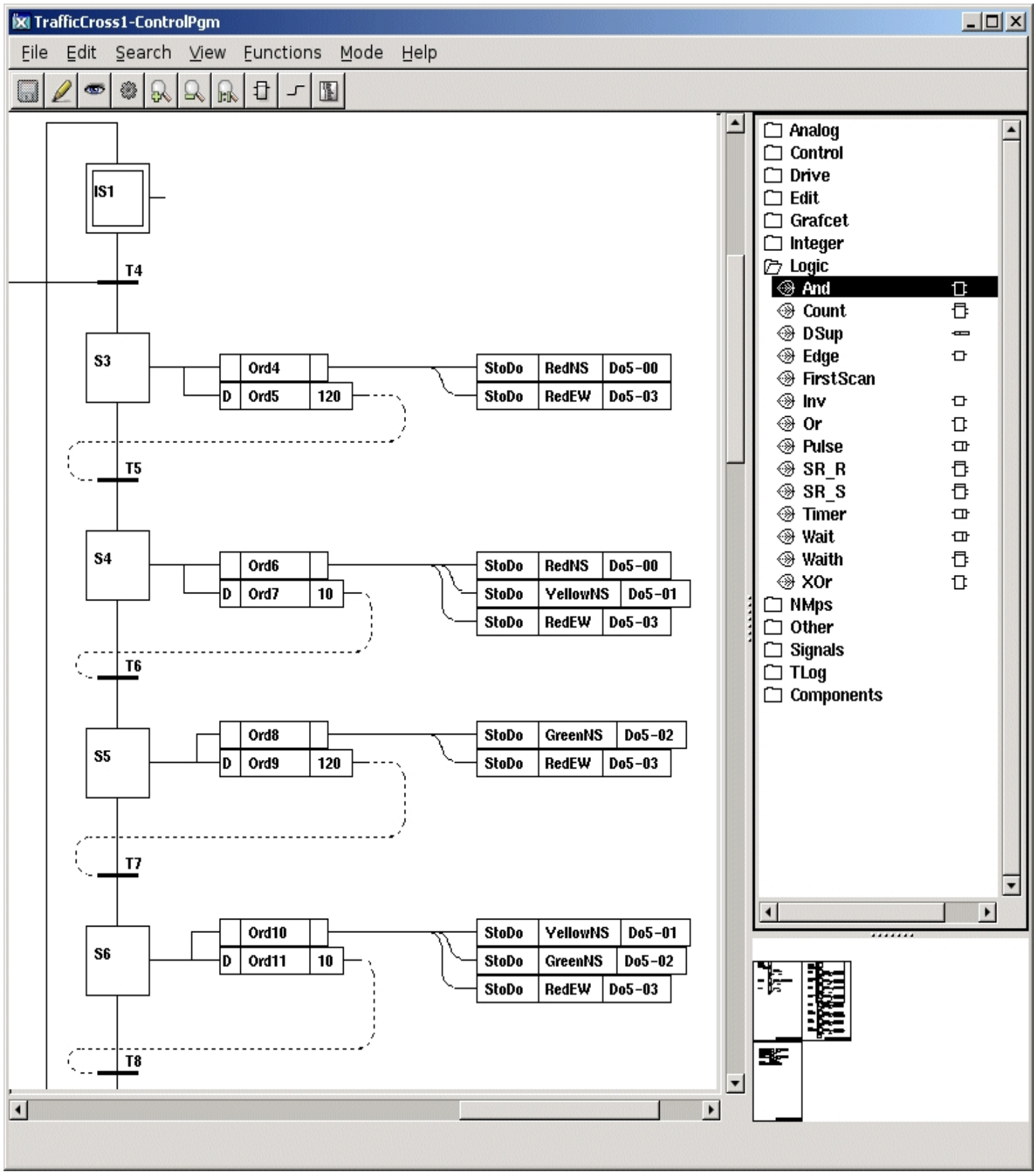
Felsökning görs numera snabbare och enklare från Xtt. PlcTrace funktionen i Xtt är att föredra framför Trace-funktionen i plceditorn.

Editering

Plc editorn består av

- en arbetsarea.
- två stycken paletter, en för funktions objekt och en för förbindningar (endast en palett i taget är synlig).
- ett navigationsfönster, från vilket arbetsarean kan skrollas och zoomas.

Plc editorn



Paletter

Objektspaletten
När man startar plceditorn, visas paletten för funktionsobjekt. När man skapar ett funktionsobjekt i arbetsarean väljer man ut en klass i paletten.

Förbindningspaletten

När man skapar förbindningar mellan objekt, väljer editorn en lämplig typ av förbindning. Men i vissa fall krävs att konstruktören påverkar valet av förbindningstyp. Detta görs i förbindningspaletten som visas genom att 'View/Palette/Connection' aktiveras i menyn. Så länge menyn är öppen, skapas alla nya kopplingar med den typ som är vald i paletten. När paletten stängs genom att 'View/Palette/Object' eller 'View/Palette/Plant' aktiveras, väljer editorn förbindningtypen igen.

Anläggningshierakin

Man kan visa anläggningshierakin genom att akivera 'View/Palette/Plant' i menyn. När man kopplar funktionsobjekt t ex för att hämta signalvärden, till signaler, kan man här markera den signal som ska hämtas. Man kan även välja ut signalen i konfiguratören, vilket i många fall är ett smidigare alternativ.

Navigationsfönster

Längs ner till vänster finns en förminskad bild av plcprogrammet. Den del av arbetsarean som visas i huvudfönstret är markerad med en rektangel. Genom att flytta på rektangeln (dra med MB1) skrollar man i huvudfönstret. Man kan även zooma genom att dra med MB2.

Funktionsobjekt

Skapa objekt

För att kunna skapa objekt måste editorn vara i editeringsmod, genom att 'Mode/Edit' aktiveras i menyn.

Ett objekt i skapas genom att man väljer ut en klass i paletten, och klickar med MB2 (mittenknappen) i arbetsarean.

Modifiera ett objekt

Ett objekt skapas med vissa defaultvärden. Det gäller även vilka in och utgångar som visas i plceditorn, och som kan förbindas med andra objekt. Vill man ändra på något värde öppnas objektseditorn för objektet. Objektseditorn öppnas genom att

- dubbelklicka på objektet.
- aktivera 'Open Object' från popupmenyn för objektet.
- välj ut objektet och aktivera 'Functions/Open Object' i menyn.

Från objektseditorn kan man mata in värden på olika attribut. Attributen för ett plc-objekt är uppdelade i ingångs-attribut, interna attribut och utgångs-attribut.

Ingångar

Ingångs-attributens värden kan hämtas från ett annat funktionsobjekt via en förbindning. Attributet visas i funktionsobjektet som en ingångspinne. I vissa fall används inte en ingång, ett and-objekt har t ex 8 stycken ingångar men ofta används inte mer än två av dessa. Detta åstadkommer man genom med hjälp av 'Used' checkboxen i objektseditorn. Om 'Used' är ifylld, visas attributet som en ingångspinne, annars inte.

En del ingångsattribut, oftast av analog typ, kan datasättas i objektseditorn. Om 'Used' inte är markerat för attributet används det datasatta värdet, om 'Used' däremot är markerat, hämtas värdet från den utgång som attributet är förbundet med. Så här fungerar t ex gränsvärdena 'Min' och 'Max' i ett Limit objekt. Man kan välja om värdet ska hämtas från ett annat funktionsobjekt, eller om det ska datasättas. Datasättningen i objekteditorn fungerar

i runtime som ett intialvärde, som sedan kan modifieras på olika sätt.

En del digitala ingångar kan inverteras. Detta görs genom att checkboxen för 'Inverted' markeras i objektseditorn. I funktionsobjektet visas detta med en rund ring på ingångspinnen.

Interna attribut
Interna attribut kan innehålla konfigureringsvärden som datasätts i utvecklingsmiljön, eller värden som beräknas i runtime. Den senare typen är ofta inte ändringsbar, och kanske inte ens synlig i utvecklingsmiljön.

Utgångar
Värdet på ett utgångs-attribut skickas vidare till en ingång med en förbindning. Liksom för ingångar, kan man välja om en utgångpinne ska visas eller ej, med 'Used' checkboxen i objektseditorn.

Välja ut objekt

Objekt väljs ut på föjande sätt

- klicka med MB1 på objektet.
- Shift/Click MB1 adderar objektet till listan av utvalda objekt, eller raderar om objektet redan är utvalt.
- genom att dra med MB1 kan man välja ut ett eller flera objekt. Objekt som har någon del inom den markerade rektangeln, väljs ut.
- genom att trycka på Shift och dra med MB1 adderar man objekt som ligger inom den markerade rektangeln till urvalslistan.

Utvalda objekt ritas med röd färg.

Flytta objekt

Enstaka objekt flyttas genom att man placerar markören på dem och drar med MB1.
Flera objekt flyttas genom att man markerar dem drar på ett av objekten med MB1.

Förbindningar

Skapa förbindningar

En utgångpinne och en ingångpinne kopplas på följande sätt

- placera markören på pinnen, eller på ett område i funktionsobjektet nära pinnen, och tryck på MB2 (mittenknappen).
- Dra markören till den andra pinnen, eller till ett område i funktionsobjektet nära pinnen, och släpp MB2.

En förbindning skapas nu mellan objekten.

Två ingångar kan kopplas ihop på samma sätt, men någon av de sammankopplade ingångarna, måste vara kopplad till en utgång, och från denna utgång hämtas värdet till samtliga sammankopplade ingångar.

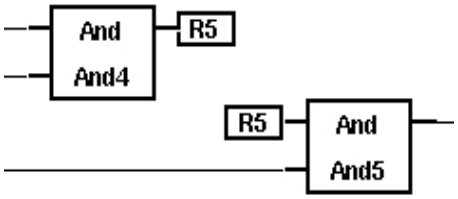
Datatyper
De värden som förflyttas mellan olika objekt via förbindningarna kan vara digitala, analog, heltals och sträng värden. In och utgångar som kopplas ihop måste vara av samma typ. Om de är av olika typ måste någon form av objekt som konverterar mellan olika typer användas, t ex AtoI eller ItoA. Dessa ligger under 'Signals/Conversion' i paletten.

Analoga- och heltalskopplingar markeras med lite tjockare linjer, digitala kopplingar med lite tunnare.

Dessutom finns det en kopplingstyp för överföring av en objektreferens. Dessa ritas med tjock streckad linje.

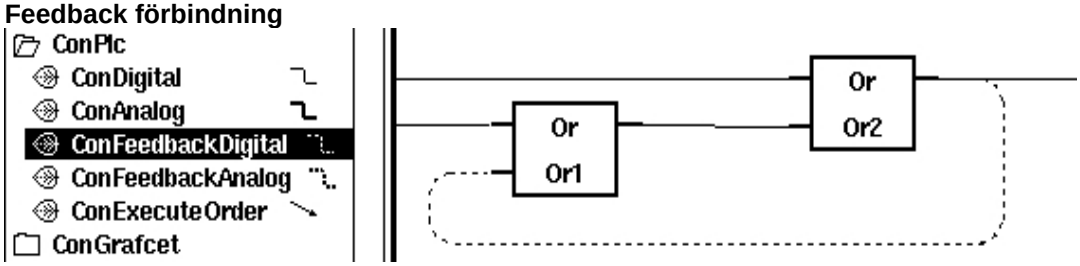
Referenskopplingar
Om editorn har svårt att hitta en väg för kopplingen mellan in och utgångspinnen, pga att det ligger många objekt ivägen, eller för att de ligger på olika dokument, ritas kopplingen som en referenskoppling. Referenskopplingar kan även ritas genom att aktivera 'View/Reference connections' i menyn.

Referens förbindning



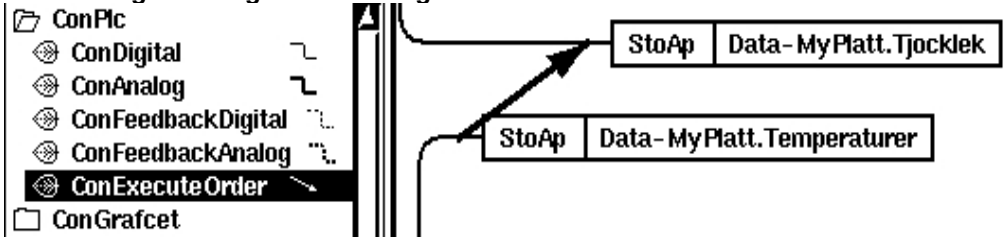
Exekveringsordning
Förutom att överföra signalvärden, bestämmer förbindningar även exekveringsordningen mellan olika funktionsobjekt. Om två objekt är sammankopplade via en utgång och en ingång, ska normalt utgångs-objektet exekveras före ingångs-objektet. Men ibland gör man en återkoppling i nätet och råkar ut för en exekverings loop. För att kunna bestämma exekveringsordingen måste man då markera återkopplingen med en koppling av typen ConFeedbackDigital eller ConFeedbackAnalog. Dessa väljs från kopplings-paletten som visas genom att 'View/Palette/Connection' aktiveras i menyn. Under mappen 'ConPlc' finns bl a feedback kopplingarna. Dessa ritas med streckade linjer.

Feedback förbindning



Här hittar man även kopplingstypen 'ConExecuteOrder'. I vissa fall vill man styra exekveringsordningen mellan två funktionsobjekt, trots att de inte är förbundna med varandra. Man kan då dra en ConExecuteOrder mellan dem (mellan vilka in eller utgångar är egalt). Kopplingen ska dras från det objekt som ska exekveras först, till det som ska exekveras sist. I figuren nedan exekveras lagringen av attributet 'Temperaturer' före lagringen av attributet 'Tjocklek'.

Exekveringsordnings förbindning



Hämta och lagra signalvärden

Hämta signal och attributvärden

I funktionsblocksnätets vänstra del hämtas värden på signaler och attribut upp. Upphämtningen av signaler sker med med objekten GetDi, GetDo, GetDv, GetIi etc. Upphämtning av attributvärden sker med GetDp, GetIp, GetAp och GetSp. Dessa objekt återfinns under mappen 'Signals' i paletten. När man har skapat ett objekt av den här typen, måste man ange vilken signal, eller vilket attribut som ska hämtas upp. Detta görs enklast genom att välja ut signalen/attributet i konfiguratören, och klicka med Ctrl/dubbelclick MB1 på objektet. Signalen/attributet visas då i objektet, och är det frågan om en insignal visas även kanalen som signalen är kopplad till.

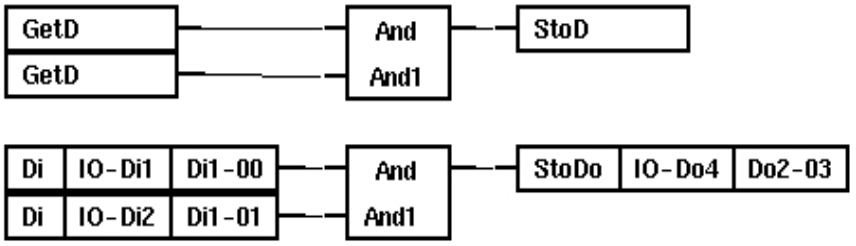
Det finns ett snabbare sätt att skapa de här objekten. Drar man ut en koppling från en ingångspinne på ett funktionsobjekt, och släpper den på ett tomt område i arbetsarean skapas ett generiskt Get objekt av den datatyp som ingången har, dvs en GetDgeneric, GetIgeneric, GetAgeneric eller GetSgeneric. När man anger den signal eller attribut som Get objektet ska hämta, omvandlas det generiska Get objektet till ett Get objekt som hämtar den typ av signal eller attribut som är angivet. Väljer jag ut en Dv i konfiguratören, kommer ett GetDgeneric att omvandlas till en GetDv när jag klickar med Ctrl/dubbelklick MB1 på den.

Lagra signal och attributvärden

I nätets högra del lagras de beräknade värdena i signaler eller attribut. Lagringen sker med objekt av typen StoDo, StoDv, StoDp, StoIo etc. Metoden för att ange vilken signal eller vilket attribut som värdet ska lagra i, är detsamma som för Get objekt, dvs genom att välja ut signalen/attributet i konfiguratören och klicka med Ctrl/dubbelklick MB1 på objektet.

Om man drar ut en koppling från en utgångspinne på ett funktionsobjekt skapas generiska Sto objekt, som omvandlas till Sto objekt av lämplig typ när de kopplas till en signal eller ett attribut. Vill man lagra värden med Set eller Reset (t ex SetDo eller ResDo) kan man inte använda den här metoden utan objektet hämtas från paletten.

Generiska Get och Sto objekt



Underfönster

Vissa objekt innehåller underfönster, t ex CSub, SubStep, Trans, Order. Att objeket har ett underfönster markeras med att vissa delar av funktionsobjektet ritas med tjock grå linje.
Ett underfönster kan öppnas på olika sätt:
- genom att välja ut objektet och aktivera 'Functions/Subwindow' i menyn.
- genom att aktivera 'Subwindow' från popupmenyn för objektet.
- genom att klicka på objektet med Shift/dubbelklick MB1.

Man skapar ett nytt underfönster på följande sätt (det faktum att man endast kan ha en

editeringssession öppen åt gången, gör skapandet en smula omständligt)

- skapa objektet som ska innehålla underfönstret.
- spara.
- öppna underfönstret.
- lämna editeringsmod i huvudfönstret.
- gå in i editeringsmode i underfönstret.

Kontrollera exekveringsordningen

I vanliga fall ska man inte behöva fundera så mycket på i vilken ordning olika funktionsobjekt i ett fönster exekverar. Genom att signaler I/O-kopieras, dvs varje tidbas i plcprogrammet, tar en kopia av alla signalvärden före exekveringen som inte förändras under exekveringen, kommer lagring och hämtning av signalvärden inte att påverkas av exekveringsordningen mellan enskilda hämtnings och lagrings objekt.

Om man däremot lagrar och hämtar värden från attribut som inte I/O-kopieras kan exekveringsordningen vara av betydelse för funktionen.

Exekveringsordningen bestäms av förbindningarna mellan funktionsobjekten. De vanliga förbindningarna är både signalöverförande och exekveringsordnings-bestämmande. Gör man en återkoppling, måste man välja en kopplingstyp som är signalöverförande, men inte exekveringsordnings-bestämmande. De olika feedback-kopplingarna är av denna typ. Dessutom finns det en typ av koppling som är exekveringsordnings-bestämmande men inte signalöverförande, ConExecuteOrder. Med den här kan man styra exekveringsordningen mellan olika funktionsobjekt utan att överföra några signalvärden.

Exekveringsordningen för funktionsobjekten i ett plcfönster visas med 'View/Show execute order' i menyn. Den siffra som visas på varje funktionsobjekt anger den ordning i vilken de exekveras. De objekt som inte har någon siffra, har inte någon exekverbar kod.

Exekveringsordningen mellan olika PlcPgm styrs av attributet ExecuteOrder i PlcPgm objektet. ExecuteOrder avgör hur exekveringen inom en plc tråd ordnas. Lägre värden på ExecuteOrder exekverar före högre värden.

Kompilering

Innan ett plc-fönster kan exekveras måste det kompileras. Samtidigt utförs en syntax kontroll av plc-koden. Om syntaxen inte är korrekt, skrivs felmeddelanden i meddelandefönstret. Felmeddelandet kan vara av typen Error eller Warning. Error är ett allvarligare fel som måste åtgärdas. Genom att klicka på pilen framför felmeddelandet i meddelandefönstret, letas det felaktiga objektet upp i plceditorn.

Efter syntaxkontrollen genereras c-kod som skickas till c-kompilatorn. Om det finns objekt med egendefinierad c-kod, t ex CArithm eller DataArithm, kan c-kompilatorn hitta fel som skrivs ut i terminalfönstret. Kontrollera alltid i terminalfönstret att kompileringen gick bra.

Kompileringen utförs genom att 'File/Build' aktiveras i menyn.

Om man vill kontrollera syntaxen utan att generera kod, aktiverar man 'File/Syntax'. Här körs inte c-kompilatorn, så eventuella c-kods fel upptäcks inte.

Klipp och klistra

Plc editorn innehåller en pastebuffer. Pastebufferten är gemensam för alla fönster, vilket gör att man kan kopiera mellan olika fönster. Med funktionerna 'Edit/Copy' och 'Edit/Cut' i menyn kopieras utvalda objekt till paste bufferten (vid Cut tas de även bort ur arbetsarean). Funktionen 'Edit/Paste' kopierar pastebufferten till arbetsarean. De kopierade objekten flyttas nu med markören, och man placerar dem på rätt plats och klickar på MB1 för att låsa fast dem.

Cut, Copy och Paste kan även aktiveras från tangentbordet med Ctrl/X, Ctrl/C och Ctrl/V.

Speciella plcobjekt

Här beskrivs ett antal objekt som har speciella funktioner i plcprogrammet.

Document
Dokumentobjekt används för att dela in koden i sidor vid utskrift. När man öppnar ett nytt fönster innehåller det ett dokumentobjekt. Via objektseditorn kan man ändra dimension på documentet, och ange signatur och sidnummer. Övriga uppgifter i dokumenthuvudet fylls i automatiskt. Dokument objekt finns under mappen 'Edit' i objektspaletten.

ShowPlcAttr
ShowPlcAttr kan användas som som en utökning av dokumenthuvudet. I detta objekt skriv info om volym, scantid och resetobjekt för Grafcet ut.

Head, Title, Text och BodyText
Dessa objekt används för att skriva förklarande texter i dokumenten. Head, Title och Text innehåller enradiga texter av olika storlek på max 79 tecken. BodyText innehåller en flerradig text på max 1023 tecken. Objekten återfinns under'Edit' i objektspaletten.

Point
Point objektet är en fri kopplingspunkt som kan användas som förgrening av förbindningar eller för att styra layouten på förbindningar. Point finns under 'Edit' i objektspaletten.

Grafcet
Grafcet sekvenser är uppbyggda med speciella Grafcet objekt som InitStep, Step, Trans och Order. Kopplingarna mellan objektet följer bestämda regler. De vertikala pinnarna på ett Step objekt ska t ex kopplas till Trans objekt, och den horisontella pinnen ska kopplas till ett Order objekt. Här följer ett exempel på hur men enkelt skapar en Grafcet sekvens.

Börja med att skapa ett InitStep objekt. Dra ut en förbindning från den undre pinnen och släpp den i arbetsarean under InitStep objektet. Nu skapas ett Trans objekt som är förbundet med InitStep objektet. Dra en koppling från Trans objektets undre koppling och släpp den i arbetsarean under Trans objektet. Här skapas nu ett Step objekt. Drar man ut en koppling från Step objektets under pinne skapas ett Trans objekt till. Vill man ha en förgrening drar man ut ytterligare en koppling från Step objektets nedre pinne. Nu skapas en förgrening med speciella StepDiv förbindningar. Om man pss skapar en förgrening från ett Trans objekt genom att dra ut två stycken kopplingar från den undre pinnen, bildas en parallellförgrening med TransDiv förbindningar som markeras med dubbla linjer.
Drar man ut en koppling från den horisontella pinnen på ett Step objekt skapas ett Order objekt osv. Som synes kan man mycket snabbt bygga upp avancerade sekvenser på det här sättet.

ScanTime
ScanTime hämtar upp den verkliga scantiden, dvs tiden sedan senaste scan.

FirstScan
FirstScan är sann första varvet plc't exekverar efter uppstart av ProviewR. Den är även sann först varvet efter en mjuk omstart.

Meny

File/Save	Spara
File/Print/Documents	Skriv ut alla dokument.
File/Print/Overview	Skriv ut en översikt på ett blad.
File/Print/Selected documents	Skriv ut utvalda dokument.
File/Syntax	Gör en syntaxkontrol av koden.
File/Compile	Kompilera programmet.
File/Plc Attributes	Öppna objektseditorn för PlcPgm objektet.
File/Delete Window	Ta bort plcfönstret.
File/Save Trace	Spara traceobjekt.
File/Restore Trace	Återskapa sparade traceobjekt.
File/Close	Stäng fönstret.
Edit/Undo Delete	Ångra borttagning av objekt.
Edit/Undo Select	Nollställ utvalslistan.
Edit/Cut	Klipp ut utvalda objekt.
Edit/Copy	Kopiera utvalda objekt till paste bufferten.
Edit/Paste	Kopiera pastebufferten till arbetsarean.
Edit/Connect	Koppla utvalt objekt till utvald signal eller attribut i konfiguratören.
Edit/Delete	Ta bort utvalda objekt.
Edit/Change Text	Ändra text i utvalt textobjekt.
Edit/Expand Object	Expandera utvalt objekt.
Edit/Compress Object	Komprimera utvalt objekt.
Search/Object	Sök på objektsnamn.
Search/String	Sök efter textsträng.
Search/Next	Sök vidare med samma textsträng.
View/Palette/Object	Visa funktionsobjekts paletten.
View/Palette/Connection	Visas förbindnings paletten.
View/Palette/Plant	Visa anläggningshierarkin.
View/Reference connections	Skapa förbindningar som referenskopplingar.
View/Grid Size	Sätt gridstorlek.
View/Show Grid	Visa griden.
View/Zoom/In	Zooma in.
View/Zoom/Out	Zomma ut.
View/Zoom/Reset	Återställ till ursprunglig zoomningsgrad.
View/Show Execute Order	Visa exekveringsordning för funktionsobjekten.
View/Redraw	Dra om förbindningar och rita om fönstret.
Functions/Open Object	Öppna objektseditorn för utvalt objekt.
Functions/Subwindow	Öppna underfönster för utvalt objekt.
Mode/View	Visningsmod.
Mode/Edit	Editeringsmod.
Mode/Trace	Felsökningsmod.
Mode/Simulate	Simuleringsmod.

Musfunktioner

Arbetsarean

Klick MB1	Välj ut objekt. Klick i tomt område nollställer utvalslistan.
Shift/Klick MB1	Addera objekt till utvalslistan.
Dubbelklick MB1	Öppna objektseditorn.
Shift+Ctrl/Dubbelklick MB1	Kopiera till pastebuffer. Klick på ett objekt kopierar objektet, klick i tomt område kopierar utvalda objekt.
Drag MB1	På objekt: flytta objekt eller flytta utvalda objekt. I tomt område: välj ut objekt inom markerad rektangel.
Shift/Drag MB1	Addera objekt inom markerad rektangel till utvalslistan.
Klick MB2	Skapa objekt.
Dubbelklick MB2	Ta bort. Klick på ett objekt tar bort objektet, klick på tomt område i arbetsarean tar bort utvalda objekt.
Shift+Ctrl/Klick MB2	Paste. Kopiera pastebufferten till arbetsarean.
Shift+Ctrl/Dubbelklick MB2	Klipp ut. Klick i objekt tar bort objektet, klick på tomt område tar bort utvalda objekt. Borttagna objekt läggs i pastebufferten.
Press MB3	Popupmeny.

Navigationsfönster

Drag MB1	Skrolla arbetsarean.
Drag MB2	Zooma arbetsarean.

22 Hjälpfil

Hjälptexter visas i hjälp fönstret, som öppnas från konfiguratören eller från operatörsmiljön. Hjälptexter skrivs i filen $pwrp_exe/xtt_help.dat. Hjälptexterna delas upp i ämnen (topics), och varje ämne har en nyckel, som specificerar när hjälptexten för ämnet ska visas. Länkar i hjälptexterna, som pekar på andra ämnen, gör det möjligt att navigera i hjälptexterna.

Ämnet 'index' är rot ämnet som visas från olika verktyg

- 'Help/Project' i konfiguratorns meny.
- 'Help/Project' i runtime navigatorns mey .
- 'Help' knappen i operatörsfönstret.

Specificerade hjälpämnen kan öppnas från Ge grafer genom tryckknappar (aktionstyp Help), eller från popup-menyn för ett objekt i operatörsmiljön (metod 'Help').

22.1 Konvertering

Hjälptexterna kan konverteras till html, PDF och PostScript format. Vid konvertering till html, konverteras varje ämne till en html sida. Vid konvertering till PDF och PostScript, finns ett antal ytterligare taggar, för att skapa dokument av hjälptexterna med kapitel och rubriker.

Konverteringen görs med 'co_convert'.

Konvertering till html

En hjälpfil konverteras till html med kommandot

```
co_convert -f [-d outputdirectory] 'helpfile'
```

Exempel

```
co_convert -f -d $pwrp_web $pwrp_exe/xtt_help.dat
```

Konvertering till PostScript

En hjälpfil konverteras till postscript med kommandot

```
co_convert -n [-d outputdirectory] 'helpfile'
```

Exempel

```
co_convert -n -d $pwrp_lis $pwrp_exe/xtt_help.dat
```

Konvertering till PDF

En hjälpfil konverteras til PDF med kommandot

```
co_convert -f [-d outputdirectory] 'helpfile'
```

Exempel

```
co_convert -f -d $pwrp_lis $pwrp_exe/xtt_help.dat
```

22.2 Encoding

Standard kodningen på hjälpfilen är ISO 8859-1. UTF-8 kan anges med en Coding specifikation på först raden, t ex

```
Coding:UTF-8
```

Tillåtna värden på Coding är UTF-8 och ISO8859-1.

22.3 Syntax

Det finns ett antal olika taggar som påverkar sökningen och konverteringen av hjälpfiler.

topic	Definierar hjälptexten för ett ämne
bookmark	Defininerar en position inne i ett ämne
link	Länk till ett ämne elller en URL
index	Lista över ämnen
h1	Rubrik 1
h2	Rubrik 2
b	Fet text
c	Kod
t	Tab
hr	Horisontell linje
include	Inkludera andra hjälpfiler

PDF och PostScript taggar

Följande taggar används för att formatera hjälptexter vid konvertering till PDF och PostScript.

chapter	Dela upp ämnen i kapitel
headerlevel	Öka eller minska rubrik nivån
pagebreak	Ny sida
option	Optioner
style	Specifik text stil

Titelsida och dokument info

Exempel

22.3.1 Topic

<topic>
<topic> börjar ett ämne och ska placeras på radens första position. Topic taggen följs av en nyckel som är sökbegrepp i hjälpfunktionen. Alla följande textrader till en </topic> tag kommer att visas som text för ämnet.

<topic> 'key'

</topic>
Avslutar ett ämne. </topic> ska placeras på radens första position.

End a topic. </topic> should bo placed in the first position
of a line.

Example
<topic> start engine
The engine will be started by...
</topic>

Kommandot

```
wtt> help start engine
```

kommer att visa texten för detta ämne.

22.3.2 Bookmark

<bookmark>
Bookmark är en rad inuti ett ämne som kan letas efter med link-taggen eller /bookmark kvalifieraren i 'help' kommandot. Bookmark ska placeras vid slutet på en rad och följas av ett namn.

'some text' <bookmark> 'name'

Exempel
This is a bookmark. <bookmark> first_engine

Kommandot

```
wtt> help start engine/bookmark=first_engine
```

kommer att visa texten för ämnet och scrolla till bokmärket.

22.3.3 Link

<link>
<link> taggen är en länk till ett annat hjålp ämne. <link> taggen ska placeras vid slutet på raden. När raden för länken aktiveras kommer ämnet för länken att visas. Link taggen ska följas av ämnet, och kan även följas av ett bokmärke, och hjälfilen som hjältexten finns i, åtskilda med kommatecken. Om en rad innehåller en länk, markeras den med en pil.

'some text' <link> 'topic'[,'bookmark'][,'helpfile']

Exempel
Link to first engine <link> show engine, first_engine

22.3.4 Index

<index>
<index> taggen är en speciell länk som visar innehållet i en hjälpfil, dvs en lista på alla ämnen i bokstavsordning.

'some text' <index>

22.3.5 Header1

<h1>
<h1> taggen kommer att visa en rad som en rubrik med större textstorlek. Taggen ska placeras i början på raden. En rubrik rad kan inte innehålla någon länk.

<h1>'header text'

Exempel
<h1>This is a h1 header
kommer att visas som

This is a h1 header

22.3.6 Header2

<h2>
<h2> taggen visar en rad som en rubrik med större textstorlek. Taggen ska placeras i början på raden. En rubrik rad kan inte innehålla någon länk.

Exempel
<h2>This is a h2 header
kommer att visas som

This is a h2 header

22.3.7 Bold

<b>
<b> taggen visar en rad med fet text. Taggen ska placeras i början på raden.

Exempel
<b>This is a bold line
kommer att visas som
This is a bold line

22.3.8 Kod

<c>
<c> taggen visar en rade med teckensnittet Courier. Taggen ska placeras i början på raden.

Exempel
<c>for (i = 0; i < 10; i++)
kommer att visas som
`for ( i = 0; i < 10; i++)`

22.3.9 Tab

<t>
<t> taggen gör det möjligt att skriva kolumner. Endast tre kolumner (två <t> taggar) är tillåtet.

Exempel
Col1 <t> Col2 <t> Col3
kommer att visas som
Col1 Col2 Col3

22.3.10 Horisonell linje

<hr>
<hr> taggen visar en horisonetell linje. Taggen ska placeras i början på raden.

Exempel
<hr>

kommer att visas som

22.3.11 Include

<include>
Inkluderar en annan hjälpfil. <include> taggen ska inte placeras inuti ett ämne.

<include> 'filname'

22.3.12 Chapter

<chapter>
Den här taggen delar in ämnen i kapitel. Ett kapitel inleds med <chapter> avslutas med </chapter>. Titeln för första ämnet i ett kapitel blir rubriken på kapitlet.

</chapter>
Avslutar ett kapitel.

Exempel
<chapter>
<topic>
Introduction
...
</topic>
</chapter>

22.3.13 Headerlevel

Delar in ämnen i ett kapitel i rubrik nivåer.

<headerlevel>
Ökar rubriknivån.

</headerlevel>
Minskar rubriknivån

22.3.14 Pagebreak

<pagebreak>
Markerar sidbrytning.

22.3.15 Option

<option>
Option kan ha följande värden

printdisable	Ignorera alla taggar och all text till nästa 'printenable' vid konvertering till PDF och PostScript. Används bl a för länkar som inte fungerar i PDF och PostScript.
printenable	Återställ 'printdisable'.

Exempel
<option> disable
Some text
...
<option> enable

22.3.16 Style

<style>
Anger att ett ämne ska skrivas med en speciell stilmall.
Stilmallar

function	Stilmall använd för funktioner och kommandon. Större rubrik och efter varje ämne.

Exempel
<topic> MyFunction <style> function
...
</topic>

22.3.17 Titelsidan och dokument info

Titelsida och sida för dokument info kan skapas med två speciella topic.

__DocumentTitlePage

Topic för titelsidan. Placeras först i en hjälp-fil.

<topic> __DocumentTitlePage
...
</topic>

__DocumentInfoPage

Topic för dokument information, t ex copyright. Placeras efter titelsidan.

<topic> __DocumentInfoPage
...
</topic>

22.3.18 Exempel på hälpfil

<topic> helpfile_example
Start and stop of engines.

Engine 1 <link> helpfile_example, bm_engine_1
Engine 2 <link> helpfile_example, bm_engine_2
Characteristics <link> helpfile_example, bm_char

<h1>Engine 1 <bookmark> bm_engine_1
Start engine one by pressing the start button.
Stop engine one by pressing the stop button.

<h1>Engine 2 <bookmark> bm_engine_2
Start engine two by pressing the start button.
Stop engine two by pressing the stop button.

<h2>Characteristics <bookmark> bm_char

<b><t>Engine1 <t>Engine2
Max speed <t> 3200 <t> 5400
Max current <t> 130 <t> 120
</topic>

Så här ser ovanstående exempel ut

22.3.18.1 Start and stop of engines.

Engine 1

Start engine one by pressing the start button.
Stop engine one by pressing the stop button.

Engine 2

Start engine two by pressing the start button.
Stop engine two by pressing the stop button.

Characteristics

	Engine1	**Engine2**
Max speed	3200	5400
Max current	130	120

23 Användare

Det här kapitlet beskriver hur man skapar användare i proview, och ger privilegier och behörighet för användaren.

Den ökade tillgängligheten på proview-system för olika typer av användare, bl a via intranätet, har gjort att kraven på att kunna begränsa olika användares möjligheter att påverka systemet ökat. ProviewR innehåller en användardatabas, där man definierar användare för olika system, och har möjlighet att gruppera system som delar användare. Databasen har designats för att möta kraven på ökad behörighets-kontroll, samtidigt som administrationen hålls på en rimlig nivå.

Användardatabasen hanteras av 'pwr_user' (se nedan) eller från administratören.

23.1 Användardatabas

Användardatabasen populeras av systemgrupper och användare. När en proview-funktion startas, t ex operatörs- eller utvecklings-miljön, kontrolleras att användaren finns i databasen och användarens privilegier registreras. Privilegerna avgör vad en användare tillåts att göra i systemet.

Systemgrupp

Begreppet systemgrupp har införts för att man inte ska behöva definiera varje system i databasen. Istället definerar man systemgrupper och knyter ett antal system till varje systemgrupp. Dessa system kommer att ha gemensamma användare.

I databasen byggs upp en hierarki av systemgrupper. Hierarkin fyller två funktioner, dels att beskriva sambandet mellan olika systemgrupper, och dels att kunna införa arv mellan systemgrupperna. Systemgrupper längre ner i hierarkin kan ärva egenskaper och användare av systemgrupper längre upp i hierarkin.

Om en systemgrupp ska ärva användare eller inte avgörs av attributet UserInherit. Om attributet är satt kommer systemgruppen att ärva samtliga användare från närmast ovanliggande systemgrupp. Även de användare som ovanliggande systemgrupp har ärvt, ärvs vidare. En systemgrupp kan överrida en ärvd användare genom att definiera användarnamnet till sin egen systemgrupp.

En systemgrupp refereras med 'path'-namnet i hierarkin där namnet är åtskilda med punkt, t ex 'ssab.hql.se1', där ssab är rot-gruppen och se1 understa nivån i hierarkin.

Ett proview-system knyts till en systemgrupp genom att systemgruppen anges i System-objektet. Om systemgruppen inte finns i användardatabasen, men någon förälder, eller förfader återfinns, antas att systemgruppen ärver användare från förfadern.

Attribut

Attribut	Beskrivning
UserInherit	Systemgruppen ärver användare av närmast ovanliggande systemgrupp i hierarkin (även system som den systemgruppen har ärvt).

Användare

En användare karakteriseras av ett användarnamn, ett passerord och en uppsättning privileger. Dessutom knyter man en användare till en systemgrupp.

Privilegierna definierar vad en användare har rätt att göra i proview. En del privilegier styr möjligheterna att kunna ändra i proview's bas-program t ex xtt eller plc-editorn, en del är avsedda att användas vid konstruktion av operatörsbilder så att man kan styra vilka inmatningsfält och vilka knappar olika operatörer ska kunna påverka.

Ett användarnamn kan vara knutet till flera systemgrupper, men ur databasen synvinkel är det olika användare, med unika passerord och privilegier. De råkar bara ha samma användarnamn.

Privilegier

Privilegier	Beskrivning
RtRead	Får läsa i rtdb. Default-privilegium utan inloggning.
RtWrite	Får skriva i rtdb. Tillåter ändring i rtdb från xtt och Simulate-mod i trace.
System	Privilegium för systemkonstruktör.
Maintenance	Privilegium för underhållstekniker.
Process	Privilegium för processtekniker.
Instrument	Privilegium för instrumenttekniker.
Operator1	Privilegium för operatör.
Operator2	Privilegium för operatör.
Operator3	Privilegium för operatör.
Operator4	Privilegium för operatör.
Operator5	Privilegium för operatör.
Operator6	Privilegium för operatör.
Operator7	Privilegium för operatör.
Operator8	Privilegium för operatör.
Operator9	Privilegium för operatör.
Operator10	Privilegium för operatör.
DevRead	Får läsa i arbetsbänken.
DevPlc	Får editera plc-program.
DevConfig	Får konfigurera arbetsbänken.
DevClass	För skapa klasser (not yet implemented)

23.2 Exempel

ProviewR user database V1.0.0

```
ssab
. . . . . . sysansv    System DevRead DevPlc DevConfig (14680068)
. . . . . . skiftel    Maintenance DevRead (2097160)
. . . . . . 55         Operator1 (64)
. hql                  UserInherit
. . . . . . anna       RtWrite Operator4 (514)
. . bl2
. . . . . . anna       Operator4 (512)
. . bl1                UserInherit
```

```
. . . . . . 55         Operator1 (64)
. . . . . . carlgustav Operator8 (8192)
. hst
. . . . . . magnus     Operator1 (64)
. . rlb                UserInherit
. . . . . . amanda     Operator4 (512)
```

Studera exemplet ovan. Det här är en listning av en användardatabas. Till vänster ser man systemgrupperna, och antalet punkter markerar deras hieraki-nivå. På samma rad står systemgruppen attribut. Under varje systemgrupp står dess användare med privilegier. Systemgruppen ssab har alltså användarna sysansv, skiftel och 55.

Systemgruppen ssab.hql.bl1 har attributet UserInherit vilket gör att den ärver användare från sin förälder. Även föräldern ssab.hql har UserInherit så ssab.hql.bl1 ärver även från ssab. Användarna för ssab.hql.bl1 blir då sysansv, skiftel, anna, 55 och carlgustav. Användaren 55 hos ssab.hql.bl1 överrider här användaren 55 hos ssab.

Systemgruppen ssab.hql.bl2 saknar UserInherit och har enbart användaren anna.

Systemgruppen ssab.hst.rlb har UserInherit och ärver från sin förälder ssab.hst. Denna har dock inte UserInherit och har inte ärvt något från sin förälder ssab. Användarna fär ssab.hst.rlb blir amanda och magnus.

Ett system med systemgruppen sandviken.hql kommer att nekas access eftersom systemgruppen och alla dess förfäder saknas.

Ett system med systemgruppen ssab.vwx.n2 kommer att ärva användare från systemgruppen ssab, dvs sysansv, skiftel och 55. Alla systemgrupper måste inte finnas i databasen, det räcker att någon förfader finns. Det som inte finns antas ha attributet UserInherit.

23.3 Inloggning

Inloggning och behörighets-kontroll fungerar olika i olika delar av proview.

Utvecklingsmiljön

När man startar navigatören öppnas ett inloggnings-fönster där man kan mata in användarnamn och passerord. Man kan även skicka med användarnamn och passerord som argument om man vill undvika inloggnings-fönstret. För att öppna konfiguratören krävs privilegiet DevRead, och för att gå in i editerings mod krävs DevRead. För att editera i plc-editorn krävs DevPlc.

Xtt-operatörsbilder

När xtt startas med ett OpPlace-objekt som argument, hämtas användaren från UserName-attributet i motsvarande User-objekt. För att få ändra i databasen från xtt krävs RtWrite. I operatörsbilderna finns olika tryck-knappar, dragreglar mm från vilka man kan påverka databasen. Dessa objekt har ett access-attribut som talar om vilka priviliger som krävs för att kunna aktivera objektet. Dessa privilegier matchas mot användarens privilegier, och om han inte har något av dem, nekas han access.

Från xtt kan man med login/logout kommandot logga in som en annan användare och därmed ändra sina privilegier.

Behöriget på web

För operatörbilder på webben finns en speciellt inloggningsfönster som kan öppnas från web menyn, om detta är konfigurerat i OpPlaceWeb objektets EnableLogin attribut. Default gruppen är common.web, som inte exsisterar i template användar-databasen. Istället används då den första existerande föräldergruppen, dvs common, och användarna i denna grupp får tillgång till hemsidan. Om man vill begränsa tillgången, skapar man antingen common.web gruppen utan UserInherit, eller en annan grupp och lägger in den i $Security objektet.
Man skapar också lämpliga användare i gruppen.

Editera användar-databasen

User-databasen editeras från administratören eller konfiguratören. Det öppnas från File/Open/UserDatabase i menyn.
Se kapitel Administration

Man kan även editera databasen med kommandon i pwr_user, se pwr_user i kapitlet Verktyg.

24 Klasseditor

Det här avsnittet beskriver hur man skapar nya klasser i ProviewR.
Det finns ett antal olika fall när det kan vara idé att skapa en ny klass.

Data objekt

Man vill lagra en mängd data i en datastruktur, t ex för att smidigare kunna få tillgång till datamängden från applikationer. Man kan även skapa dataobjekt som beskriver material som passerar genom en anläggning, där ett dataobjekt innehåller egenskaper för ett material, t ex längd, bredd, vikt etc. Material objekt kan flyttas runt i NMps celler för att beskriva läget av ett material i anläggningen, och låta detta styra processen.

Plc funktionsobjekt

Funktionsobjekt som används i plc programmeringen består av en klass som definierar in- och utgångs pinnar på funktionsobjektet samt eventuella interna attribut. Det här typen av objekt består även av kod, som exekveras av plcprogrammet. Man kan välja att skapa koden i form av plc-kod eller c-kod.

Komponenter

Ett komponent-objekt speglar en komponent i anläggningen och är ofta uppdelad i två eller tre olika klasser, ett huvudobjekt, ett funktionsobjekt och ett bussobjekt, ibland även ett simuleringsobjekt. Huvudobjektet läggs i anläggningshierarkin och innehåller de signaler som är kopplade till komponenten, jämte andra konfigureringsdata. I ett plcprogram läggs ett funktionsobjekt som kopplas till huvudobjektet och som arbetar dels med data från sina egna ingångar, och dels med signaler och andra parametrar som finns i huvudobjektet. Om signalutbytet med komponenten sker via Profibus, kan man även skapa ett speciellt Profibus modulobjekt som innehåller kanalobjekt för de data som transporteras på Profbus. Här räcker det med att göra en koppling mellan huvudobjekt och modulobjekt, för att koppla ihop alla kanaler och signaler. Simuleringsobjektet är ett funktionsobjekt som kopplas till huvudobjektet och som simulerar komponenten när systemet körs i simuleringsmod.

Subklasser av komponenter

ProviewR innehåller ett antal baskomponent-klasser för ventiler, motorer mm. Dessa är byggda mycket generalla för att täcka in ett stort antal komponenter. Ofta gör man en subklass som är anpassad till en specifik komponent, och som t ex innehåller länk till datablad, hjälptext mm. för just denna komponent. Genom att göra en subklass av en baskomponent ärver man alla metoder och attribut från denna, men har även möjligheten att utöka funktionaliteten med fler attribut och mer plc-kod.

Aggregat

Ett aggregat speglar ett anläggningsdel som består av ett antal komponenter. Här kan man göra en aggregats-klass som innehåller de olika komponenterna i form av attributobjekt.
Till aggregatet finns även ett funktionsobjekt som anropar funktionsobjekten för ingående komponenter. Aggregat kan även inehålla andra aggregat och ge upphov till ganska omfattande objektsstrukturer. I princip skulle man kunna bygga en anläggning i form att ett enda aggregatsobjekt, men praktiken är det lämpligt att hålla objektsstrukturen på en ganska låg nivå. Det är framför allt när man har flera identiska aggregat som man har nytta av att göra

ett aggregatsobjekt av anläggningsdelen.

24.1 Databasstruktur

Objekt

I avsnittet Databas struktur ges en beskrivning av hur objekt är uppbyggda. Här finns anledning att gå lite djupare i det ämnet.

Ett objekt består av ett objektshuvud och en objektskropp. Objekshuvudet innehåller information om objektets namn, klass och relation till andra objekt. Objektskroppen innehåller objektets datamängd.

Objektshuvud

Ett objekt har ett namn på maximalt 31 tecken som finns lagrat i objekthuvudet.

I objektshuvudet finns även en länk till objekts klassbeskrivning. I klassbeskrivningen finns information som gör att man kan tolka objektets datamängd, hur den är uppdela i olika attribut och vilken typ de olika attributen har. Vidare finns här även de olika metoder som kan verka på objektet.

Ett objekt ligger i en trädstruktur och i objektshuvudet finns pekare till närmsta anhöriga: förälder, föregående syskon, nästa syskon och första barn.

Strukturen på ett objekthuvud är gemensamt för alla olika typer av objekt.

Objektskropp

Ett objekt kan ha två olika kroppar, en kropp som innehåller den datamänd som behövs i runtime, dessutom kan ett objekt ha ytterligare en kropp med en datamängd som enbart finns i utvecklingsmiljön.

En kropp är uppdelad i attribut som innehåller data av en viss typ, det kan t ex vara en Boolean, Float32 eller Int32. Men ett attribut kan även vara en mer komplex datatyp, som en vektor eller en klass.

RtBody
RtBody är den kropp som finns i runtimedatabasen. Kroppen finns även i utvecklingsmiljön så att man där kan datasätta olika attribut i kroppen.

DevBody
Vissa objekt har även en DevBody, en kropp som enbart finns i utvecklingsdatabas, och som inte laddas in i runtimedatabasen. Detta används främst av plcobjekt, där devbody t ex innehåller grafisk data för plceditorn.

24.2 Klassbeskrivning

Hur ett objekts kropp ser ut finns beskrivet i objektets klassbeskrivning. Här finns även de metoder som kan verka på ett objekt beskrivna och alla övriga egenskaper hos instanser av klassen. Klassbeskrivningen byggs upp av speciella klassdefinitionsobjekt som ligger i en klassvolym. Klassvolymen har en strikt syntax över hur klassbeskrivningarna ska var uppbyggda. Här följer en presentation av de olika objekt som ingår in en klassbeskrivning.

Klassvolym

Klassbeskrivningar ligger i en speciell typ av volym, ClassVolume. Dessa kan innehålla två olika hierarkier, en hieraki med klassbeskrivningar och en med typbeskrivningar.

$ClassHier

Klassbeskrivningar ligger under rotobjektet 'Class' av klassen $ClassHier. Under $ClassHier objektet ligger objekt som beskriver olika klasser i form av $ClassDef objekt.

$ClassDef

Ett $ClassDef objekt med underliggande objekt beskriver en klass. Namnet på objeketet ger namnet på klassen. Under $ClassDef objektet kan det finnas

- ett $ObjBodyDef objekt, 'RtBody', som beskriver runtimekroppen.
- ett $ObjBodyDef objekt, 'DevBody', som beskriver kroppen i utvecklingsmiljön.
- ett Template objekt, dvs ett objekt av den aktuella klassen som innehåller defaultvärden för instanser av klassen.
- ett eller flera body objekt som innehåller data för specifika funktioner.
- ett PlcTemplate objekt, som kan öppnas av plceditorn, och som innehåller plc-kod för klassen.
- menyobjekt som beskriver popupmenyn i navigatören, konfiguratören och xtt.
- metodobjekt som knyter till metoder som anropas t ex när objekt skapas eller flyttas i utvecklingsmiljön.

$ObjBodyDef

Ett $ObjBodyDef kan ha antingen namnet 'RtBody' och beskriver då runtimekroppen, eller namnet 'DevBody' som bekriver utvecklingsmiljö kroppen. I attributet 'StructName' ligger namnet på c-structen för klassen i den include-fil som genereras för volymen. Under $ObjBodyDef objektet ligger ett objekt för varje attribut som finns i objektskroppen. För dataobjekt använder man $Attribute objekt, för funktionsobjekt $Input, $Output och $Intern.

$Attribute

Ett $Attribute objekt beskriver ett attribut i en kropp. Attributet kan vara av följande typ:
- en bastyp, t ex Boolean, Float32, Time, Int16.
- en härledd typ, t ex String80, Text1024, URL.
- en vektor av bastyp eller härledd typ.
- en annan klass.
- en vektor av en klass.
- en rtdb pekare, dvs en pekare som kan tolkas av alla processer.
- en privat pekare, dvs en pekare som endast kan tolkas av en process.

Typen anges i attributet 'TypeRef'. I attributet 'Flags' anges om objektet beskriver en vektor, pekare, klass mm. Om objektet beskriver en vektor anges antalet element i 'Elements'.

$Input

$Input beskriver en ingång till ett funktionsobjekt i plcprogrammet. Ingången kan vara av

typen Boolean, Float32, Int32, String80, Time, DeltaTime eller av datatyp (pekare till Float32). $Input ger upphov till ett attribut med två element, ett element av den angivna typen, och ett element med en pekare till den angivna typen. Om ingången är kopplad pekar pekaren på det kopplade utgångsattributet, om ingången ej är kopplad pekare den på sitt första element, där man kan då kan datasätta ett värde på ingången.

Attributet 'PgmName' anger namnet i c-structen för attributet, och 'GraphName' den textsträng som skrivs i funktionsobjektet vi ingången.

$Intern

Ger uphov till ett internt attribut i ett funktionsobjekt, dvs ett attribut som varken är en ingång eller utgång.

$Output

$Output beskriver en utgång i ett funktionsobjekt. Samma datatyper gäller för $Output som för $Input.

$Buffer

$Buffer skapar ett attribut som innehåller en datamängd av en viss storlek som endast någon enstaka funktion behöver kunna tolka. Datamängden beskrivs av en klass, men går ej att öppna i t ex xtt. PlcNode som återfinns i alla plc objekt är ett exempel på $Buffer. Där återfinns grafiska information som enbart är av intresse för plc-editorn.

Klass kropp

Ett klass kan innehålla ett klass kropp objekt. Klass kroppobjektet innehåller data som är gemensam för alla instanser av klassen. Exempel på klass kropp objekt är $GraphPlcNode som återfinns i all plc-klasser. $GraphPlcNode innehåller data för kodgenerering och grafisk layout av funktionsobjektet.

Menyer

Menyobjekt används för att definiera popup-menyer för objekt i utvecklingsmiljön och i operatörsmiljön. $Menu definierar en popupmeny i utvecklingsmiljön, och $RtMenu in operatörsmiljön. Under menuobjektet definieras menyalternativ med $MenuButton objekt, och undermenyer med $MenuCascade objekt. Meny-objekten läggs under $ClassDef objektet.

Menyobjekten anropar metoder, dvs c-funktioner som byggs med utvecklingsmiljön resp operatörsmiljön. Det finns fn inte någon möjliget att göra detta från ett projekt, utan bygget måste ske från källkodsträdet.

$Menu

$Menu objekt beskriver popupmenyer i utvecklingsmiljön. Objektsnamnet specificerar funktionen, första delen anger verktyget (Navigator/Configurator). De fem sista bokstäverna bestämmer under vilka vilkor menyn är närvarande, beroende på vilka objekt som är utvale eller utpekade.

1. P står för pointed, dvs det objekt som markören pekar på.
2. anger vad pointed är: 'o' ett objekt, 'a' ett attribut, 'c' en klass i paletten.
3. s står för selected, dvs det objekt som är utvalt.
4. anger vad selected är: 'o' ett objekt, 'a' ett attribute, 'c' en klass i paletten, 'm' flera objekt utvalda, 'n' inget objekt utvalt.
5. anger om selected och pointed är samma objekt: 's' samma objekt, 'n' olika objekt.

Exempel ConfiguratorPosos: 'Po' markören pekar på ett objekt, 'so' ett objekt är utvalt, 's' det objekt markören pekar på och det utvalda är samma objekt.

$RtMenu

Menyobjekt som beskriver popupmenyer i operatörsmiljön.

$MenuButton

Definerar ett menyalternativ i en popupmeny.

24.3 Typbeskrivning

Typbeskrivningar ligger liksom klassbeskrivningar i en klassvolym. De är placerade i en egen hierarki under ett $TypeHier objekt. Typer är uppdelade i två kategorier, bastyper och härledda typer.

Bastyper

Bastyperna ligger definierade i systemvolymen pwrs. Exempel på bastyper är Boolean, Float32 Int32, String, Enum och Mask.

Härledda typer

Härledda typer kan definieras i vilken klassvolym som helst. De utgörs av
- vektorer av bastyper, t ex String80.
- uppräkningstyper, Enum, med definerade texter för olika värden.
- bitmasker, Mask, med definerade texter för olika bitar.

$TypeHier
Typbeskrivningar ligger under rotobjektet 'Type' av klassen $TypeHier. $TypeHier objektet har $Type och $TypeDef objekt som barn.

$Type
Beskrivning av en bastyp. Detta objekt är reserverat för systemvolymen pwrs.

$TypeDef
Beskrivning av en härledd typ. Attributet 'TypeRef' innehåller bastypen. Den vanligaste användningen är strängar och texter med specifika längder, och uppräkningstyper och bitmaskar.

För en uppräkningstyp ska bastypen vara $Enum. Under $TypeDef objektet definerar man texter för olika värden med $Value objekt. När ett värdet av attribut av typen ska visas, visas den text som motsvara aktuellt värde. När attributet ska datasättas visas de olika de olika texterna med checkboxar och man väljer ut ett alternativ.

För bitmaskar används bastypen $Mask. Under $TypeDef objektet defineras texter för olika bitar med $Bit objekt. Vid datasättning väljer man liksom för uppräkningstyper, alternativ med checkboxar. För bitmaskar kan man välja flera alternativ.

$Value
Används för att definiera ett värde i en uppräkningstyp. Värdet kopplas till ett text som visas i konfiguratören och xtt när ett attribut av typen öppnas. I include-filen för volymen skapas en enum deklaration som kan användas i eventuell c-kod.

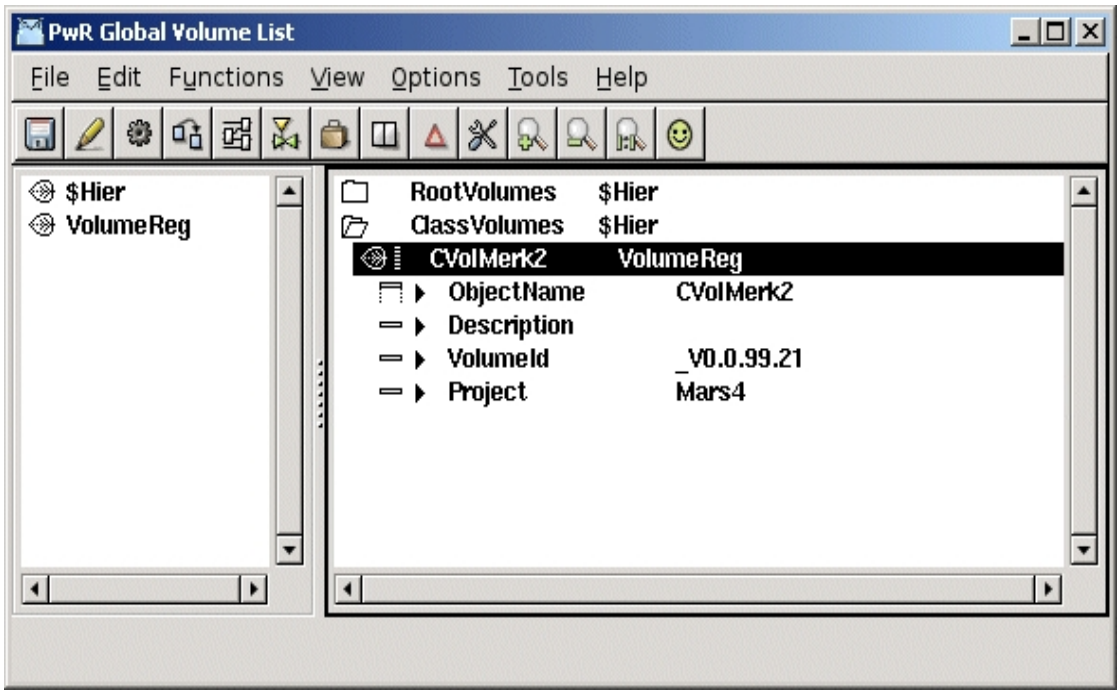
$Bit
Används för att definiera en bit i en bitmask. Biten kopplas till ett text som visas i konfiguratören och xtt när ett attribut av typen öppnas. I include-filen för volymen skapas en enum deklaration som kan användas i eventuell c-kod.

24.4 Skapa klasser

24.4.1 Skapa en klassvolym

Klassdefinition objekten ligger i en klassvolym, och först måste klassvolymen registreras och skapas.

Registreringen sker i globala volymslistan som öppnas från File/Open/GlobalVolumeList i navigatorns meny. Här skapar man ett VolumeReg objekt med lämpligt volymsnamn och volymsidentitet. Volymsidentiteten för användar-klassvolymer ska ligga i intervallet 0.0.2-249.1-254. Använd gärna prefixet CVol i namnet för att makera att det är en klassvolym. Ange även aktuellt projekt.

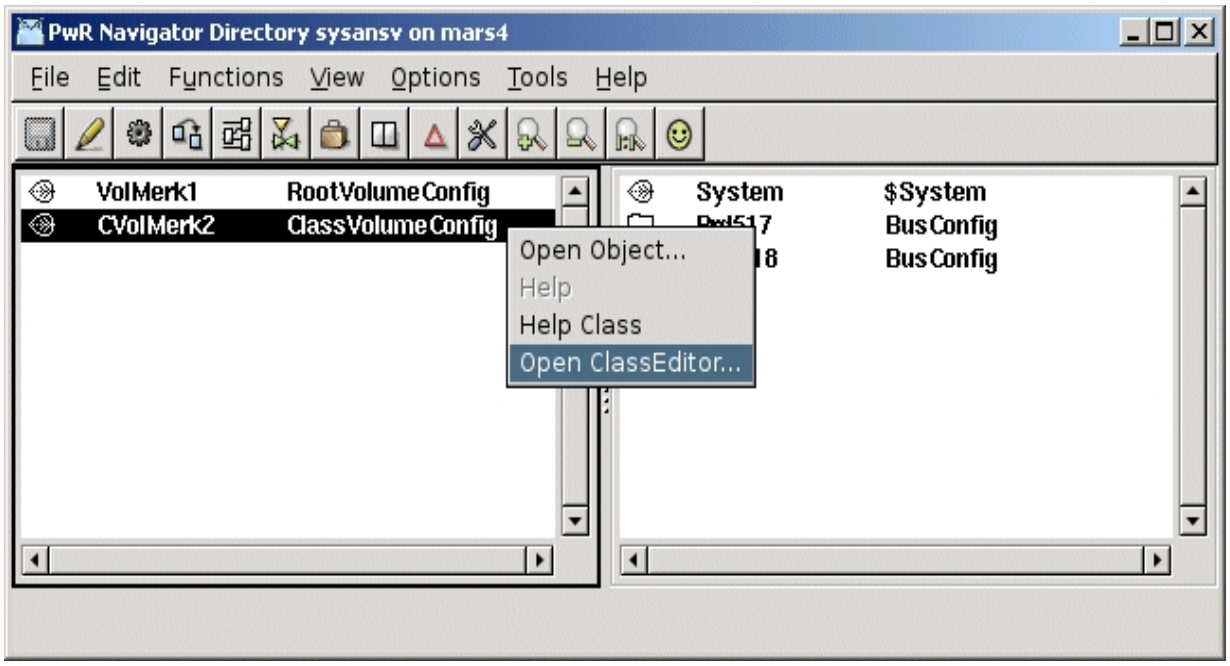


Registrering av klassvolymen i GlobalVolumeList

Därefter ska klassvolymen konfigureras i projektets directory volym med ett ClassVolumeConfig objekt. Öppna Directory volymen med

```
$ pwrs
```

och lägg ett ClassVolumeConfig objekt i det högra fönstret. Objektet ska ha samma namn som klassvolymen. När man har sparat och lämnat edit mod, kan man öppna klassvolymen genom att högerklicka på ClassVolumeConfig objektet och aktivera 'Open ClassEditor...'.



Konfiguration av klassvolymen i Directory volymen

Nu öppnas klasseditorn, där man kan skapa klassdefinitions objekt. Genom att gå in i editerings mod visas en palett, med de klass och typ definitions klasser som används för att bygga en klass eller typ.

Börja med att skapa ett objekt av typen $ClassHier på rotnivå. Det får automatiska namnet 'Class'. Under $ClassHier objektet lägger man sedan ett $ClassDef objekt för varje klass som ska defineras.

24.4.2 Data klasser

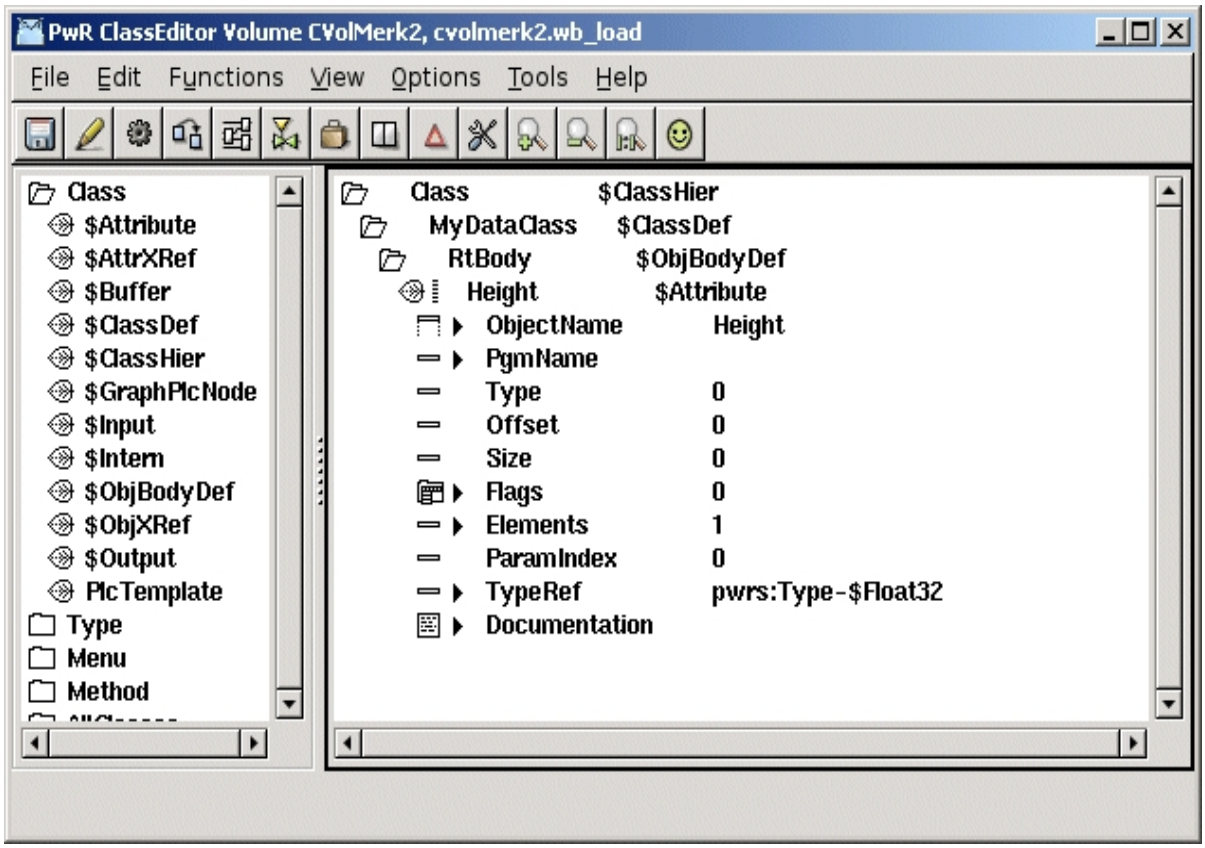
Data klasser är den enklaste typen av klasser, och används normalt för att lagra data i. Klasserna består av en RtBody med attribut.

För att skapa en klass lägger man ett $ClassDef objekt under 'Class' objektet.
Namnet på $ClassDef objektet kommer att bli klassens namn.

Under $ClassDef objektet skapar man ett $ObjBodyDef objekt som automatiskt får namnet RtBody.

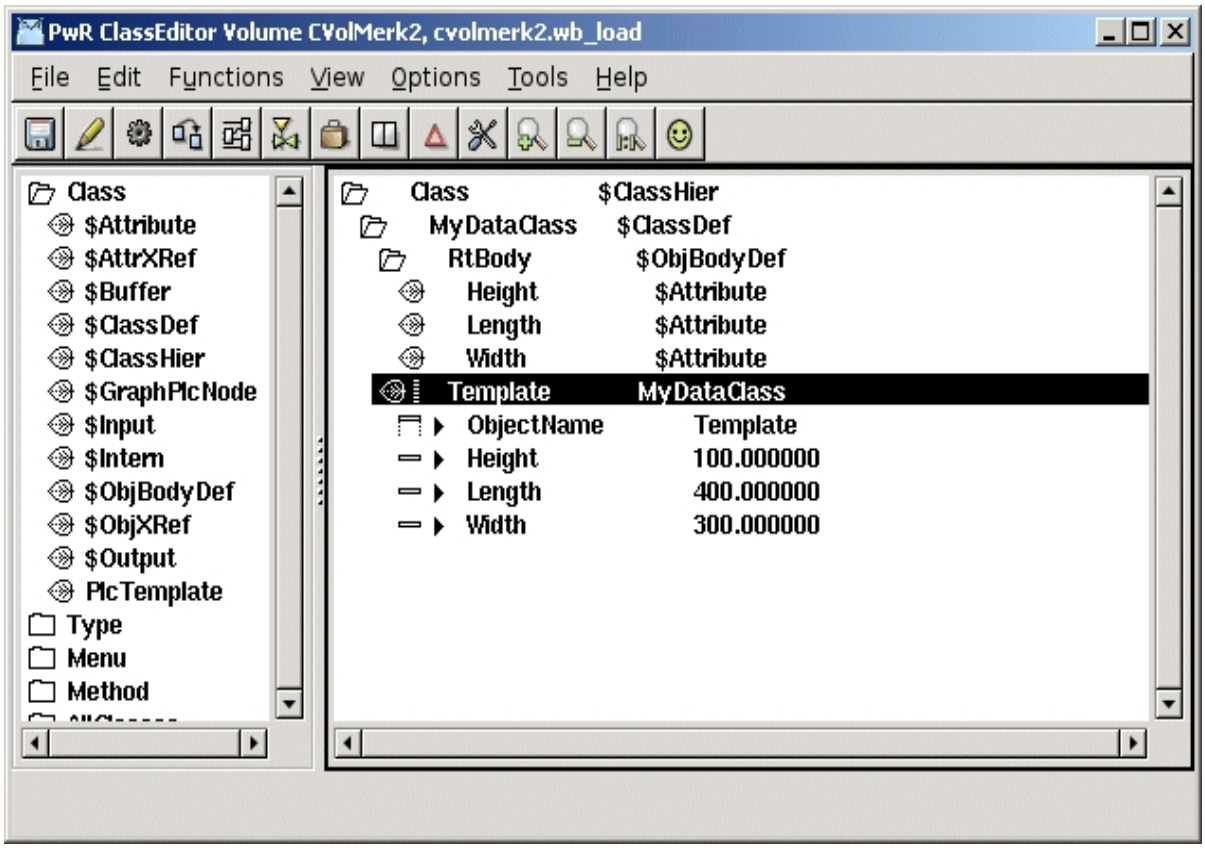
Under RtBody objektet skapas ett $Attribute objekt som definierar ett attribut i klassen.
Namnet på $Attribut-objektet ger attribut-namnet. I objektet måste anges följande:

- attributets typ anges i TypeRef. Ett 32-bitars heltal anges t ex med pwrs:Type-$Int32, ett 32-bitars flyttal med pwrs:Type-$Float32 och en boolean med pwrs:Type-$Boolean. Det som läggs in är egentligen namnet på ett typdefinitions-objekt. Se i objekts handboken pwrs/Types, vilka typer som finns definierade.
- om attributnamnet innehåller nationella tecken måste man ange ett namn utan nationella tecken som godkänns av c-kompilatorn. Detta anges i PgmName.



Definition av ett attribut

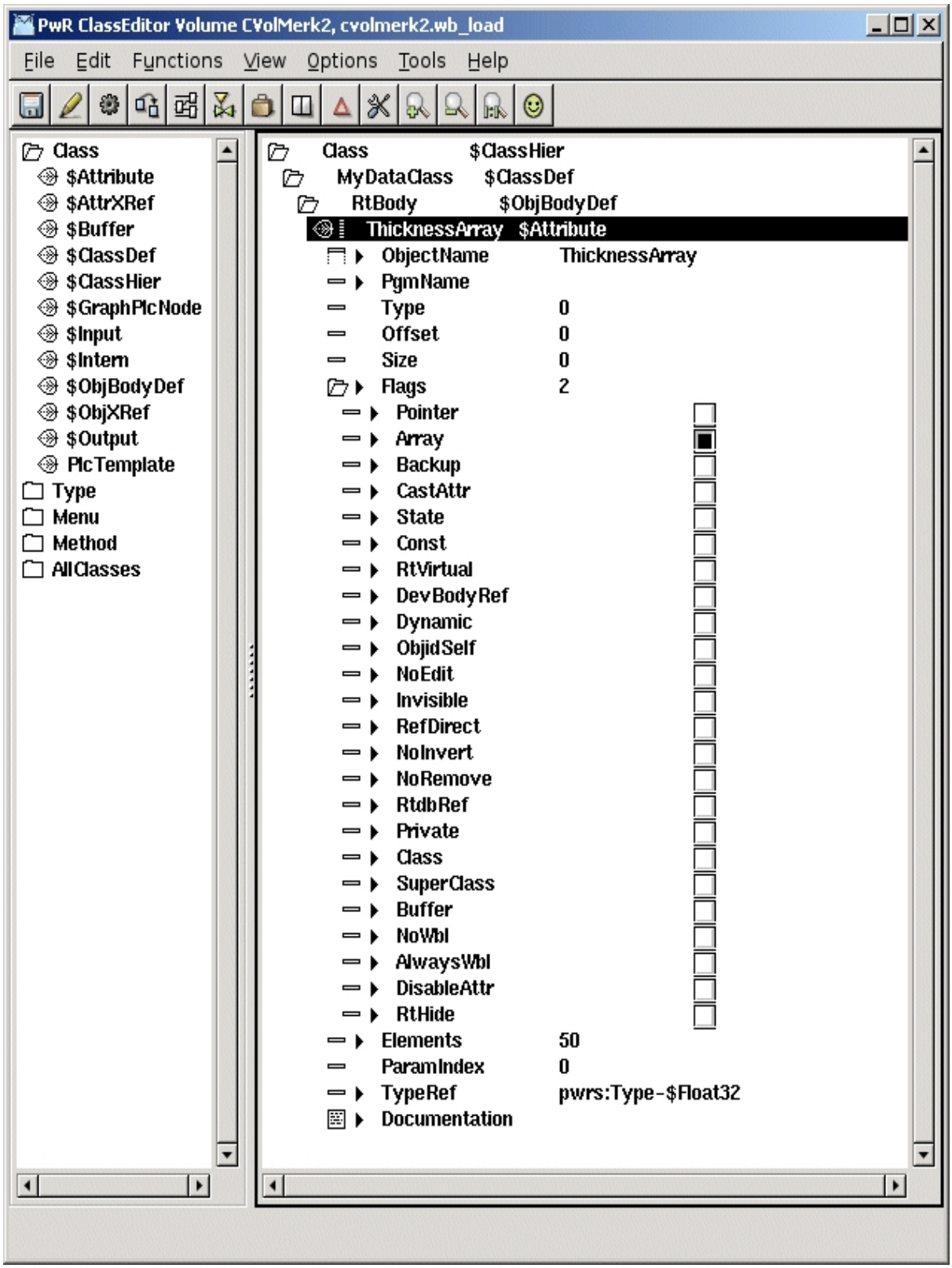
När man sparar, skapas en instans av den aktuella klassen med namnet Template under $ClassDef objektet. Här kan man se hur ett objekt av klassen ser ut. I Template objektet kan man också lägga in default-värden på attributen. När instanser av klassen skapas, tas en kopia av Template objektet.



Template objekt med defaultvärden

Vektorer

Ett vektor-attribut definieras med ett $Attribute objekt på samma som övriga attribut. Här sätter man Flags biten Array, och anger antalet element i vektorn i Elements.



Definition av vektor attribut med 50 element

Attributobjekt

Med attributobjekt avses attribut som beskrivs av en datastruktur. Orsaken kan vara att man vill samla ett antal data i objektet under en mapp, eller att datastrukturen upprepas, i det här fallet gör man ett attribut objekt i form av en vektor.

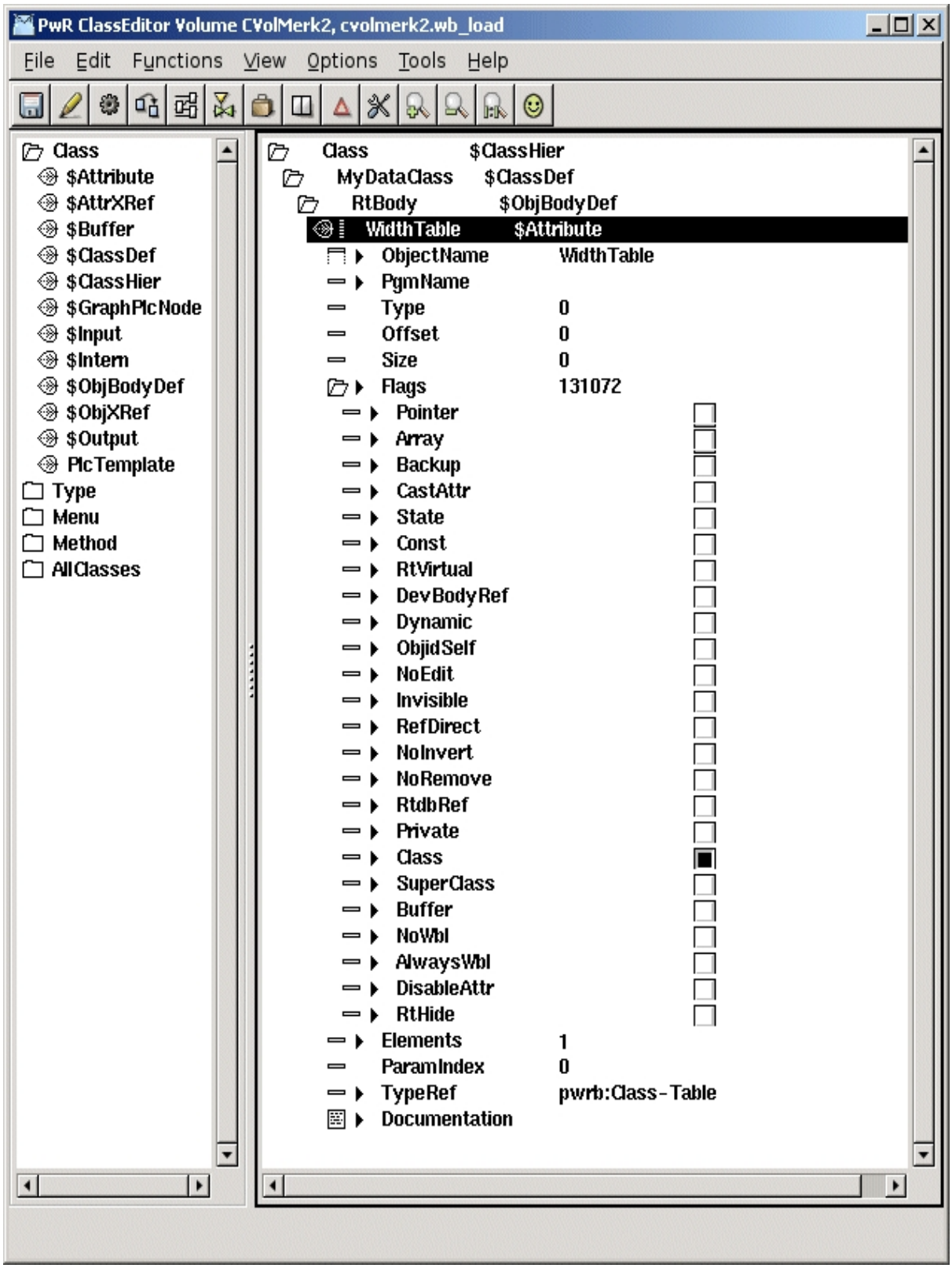
Datastrukturen för attributet måste defineras av en egen klass. Klassen ska enbart innehålla en runtime body, och får inte ha en development body.

Attributobjektet definieras med ett $Attribute objekt. Som TypeRef anges den klass som beskriver datastrukturen, och i Flags sätts biten Class.

Man kan även göra en array av attributet genom att sätta Array biten i Flags, och ange antalet element i Elements.

Attributobjekt kan i sin tur ha attribut som är attribut objekt. Antalet nivåer är dock begränsat till 20, och längden på det sammanlagda attribut-namnet får maximalt vara 255 tecken.

Ett attribut i ett attributobjekt refereras med punkt som avgränsare, dvs attributet Description i attributobjektet Pump i objektet o, refereras med namnet 'o.Pump.Description'. Om pump dessutom är en vektor av pumpobjekt blir namet på Description attributet i det första pumpobjektet 'o.Pump[0].Description'.



Definition av ett attributobjekt av klassen Table.

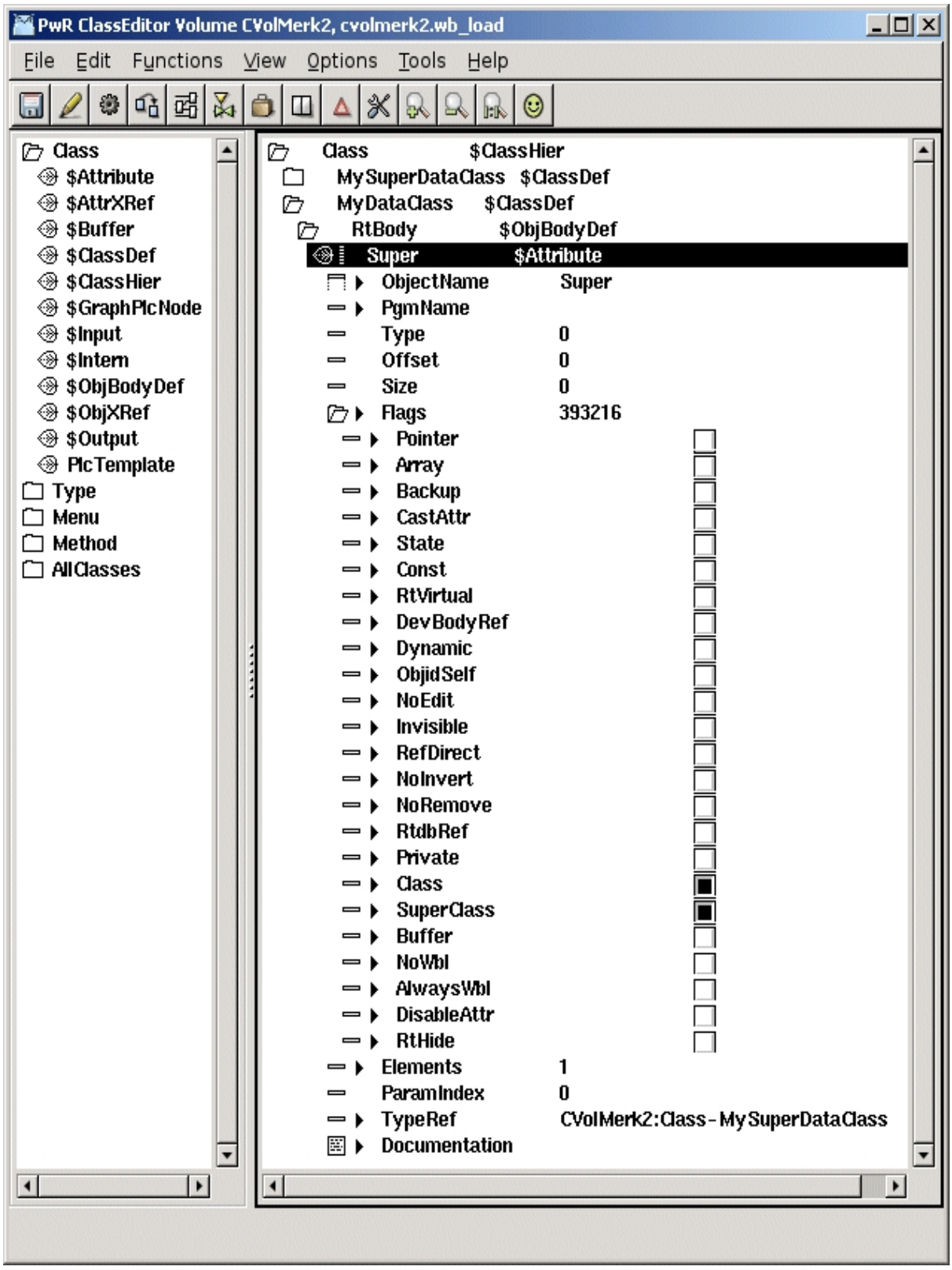
Subklass

Man kan även definiera en klass som en subklass till en annan klass. Subklassen kommer att ärva attribut och metoder av den andra klassen, som kallas för superklass.

En subklass definieras genom att det första $Attribute objektet i klassen har namnet 'Super', och bitarna Class och SuperClass satta i Flags ordet. Superklassen anges i TypeRef.

Alla attribut som finns i superklassen kommer även att finnas i subklassen. Subklassen kan byggas ut med fler attribut som defineras på normalt sätt med $Attribute objekt.

En superklass får enbart innehålla en runtime body, ej någon dev body.



Super attributet gör MyDataClass till en subklass till MySuperDataClass

24.4.3 Funktionsobjekts klasser

Funktionsobjekt används i plc editorn för att programmera plc-program. Även ett funktionsobjekt beskrivs av en klass, vanligvis lite med komplex än en data-klass, eftersom den, förutom en datastruktur, även ska definera ett grafiskt utseende med in och utgångar, och den kod som ska exekveras av plc-programmet.

Koden kan defineras antingen med c-kod, eller med grafiskt programmering i plc-editorn.

24.4.3.1 Funktionsobjekt med c-kod

Funktionsobjekts-klassen defineras med ett $ClassDef objekt under 'Class' objektet. Namnge objektet och aktivera 'Configure-CCodeFo' från popup-menyn för objektet. Nu skapas

- ett RtBody objekt.
- ett DevBody objekt med ett PlcNode objekt som definierar en buffer för grafisk information i instanserna.
- ett GraphPlcNode objekt som innehåller diverse information om grafik och kodgenerering för klassen.

Nästa steg är att definiera attribut för klassen. Attributen indelas i ingångar, interna attribut och utgångar.

Ingångar
Ingångsattributen definierar ingångarna på funktions-objektet, dvs värden som hämtas från utgångar på andra funktionsobjekt. Ingångarna definieras med $Input objekt som läggs under RtBody objektet.

I TypeRef anges datatypen för ingången, de datatyper man kan välja är pwrs:Type-$Boolean, pwrs:Type-$Float32, pwrs:Type-Int32 eller pwrs:Type-String80.

I GraphName anges texten på ingången i funktionsobjektet, normalt brukar man använda 2 - 4 bokstäver. Man brukar använda stora bokstäver för analoga signaler, små för digitala och stor första bokstav för övriga signaltyper.

Ett ingångsattribut i en instans innehåller både en pekare till den utgång den är kopplad till, och ett värde som kan datasättas. Det gör att man kan välja om en ingång ska kopplas, eller datasättas. Det här valet görs med en checkbox (Used), och om man väljer att inte markera Used, visas inte ingångs pinnen på funktionsobjektet. I Template objektet kan man sätta defaultvärde på ingången, i det fall ingången inte kopplas.

Interna attribut
Interna attribut är attribut som ej är ut eller ingångar. Det kan vara beräknade värden som behöver lagras i objektet, eller värden som används för att konfigurera objektet.

Alla vanliga datatyper är tillgängliga för interna attribut.

Utgångar
Utgångsattributen definierar utgångarna på funktions-objektet, dvs värden som lagras i objeketet, och som därifrån kan hämtas till ingångar på andra funktionsobjekt. Utgångarna definieras med $Output objekt som läggs under RtBody objektet.

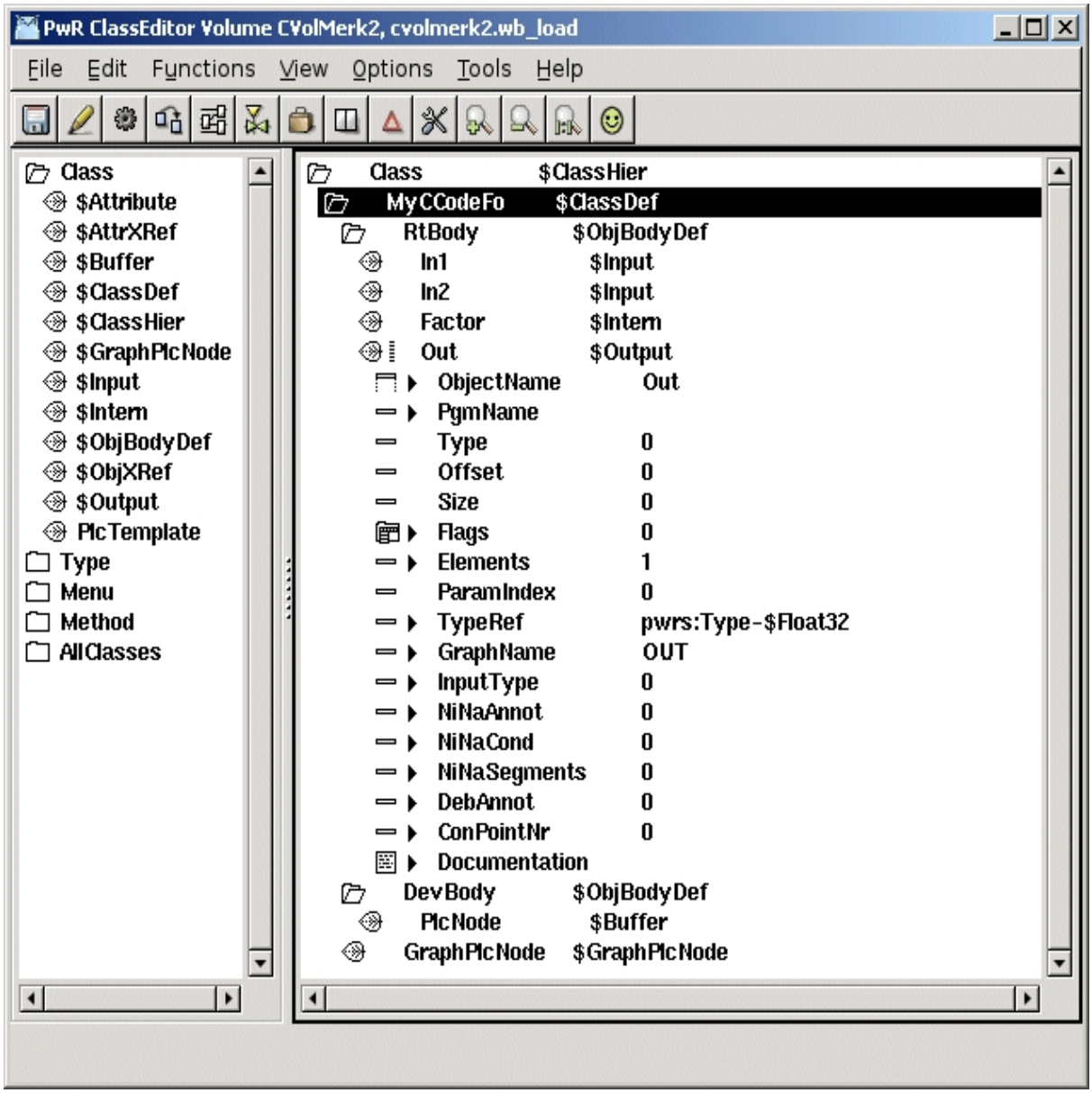
I TypeRef anges datatypen för utgången, liksom för $Input objekt kan Boolean, Float32, Int32

och String80 anges, och i GraphName anges texten vid utgångspinnen i funktionsobjektet.

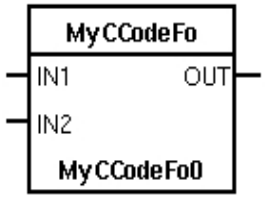
OBS !
$Input, $Intern och $Output måste ligga i följande ordning under RtBody: $Input först, sedan $Intern och därefter $Output.

Defaultvärden
Defaultvärden på attribut kan anges i Template objektet.

Om man vill ange vilka in och utgångar som ska visas som default, finns det en mask i GraphPlcNode objektet, default_mask. default_mask[0] anger ingångar och default_mask[1] utgångar. Om biten som motsvarar en in eller utgång sätts visas denna som default.



Funktionsobjekt med två ingångar, ett internt attribut, och en utgång



Funktionsobjektet för klassen

Kod
När klassvolymen byggs genereras en h-fil med en c-struct för klassen. Namnet på structen blir

```
pwr_sClass_'StructName'
```

där StructName hämtas från StructName attributet i RtBody. Default är det samma som klassnamnet, men t ex om klassnamnet innehåller nationella tecken, kan ett annat namn anges.

Här visas ett exempel på structen för klassen MyFo som innehåller två ingångar In1 och In2, ett internt attribut Factor, och en utgång Out, alla av typen Float32.

```
typedef struct {
  pwr_tFloat32           *In1P;
  pwr_tFloat32           In1;
  pwr_tFloat32           *In2P;
  pwr_tFloat32           In2;
  pwr_tFloat32           Factor;
  pwr_tFloat32           Out;
} pwr_sClass_MyFo;
```

Notera att varje ingång består av två element, en pekare med suffixet 'P', och ett element som kan datasättas om ingången inte kopplas. Om ingången är kopplad kommer pekar-elementet att peka på den utgång den är kopplad till, annars pekar den på datasättnings-elementet. I koden ska man därför använda pekar-elementet för att hämta upp värde på ingången.

Koden för klassen ska vara en funktion med följande utseende

```
void 'StructName'_exec( plc_sThread *tp,
                        pwr_sClass_'StructName' *o) {
}
```

I koden hämtas data från ingångarna, och beräknade värden läggs ut på utgångarna. Även interna attribut kan användas för att lagra information till nästa scan, eller för att hämta upp konfigureringsdata.

I kod exemplet nedan är In1 och In2 ingångar, Factor ett internt attribut och Out en utgång.

```
  o->Out = o->Factor * (*o->In1P + *o->In2P);
```

Observera att pekar-elementet för ingångarna In1 och In2 används i koden.

En prototyp-deklaration av exec funktionen läggs in i ra_plc_user.h

```
void 'StructName'_exec( plc_sThread *tp,
                        pwr_sClass_'StructName' *o);
```

Modulen för c-koden kompileras och länkas ihop med plc-programmet genom att en länk-fil skapas på katalogen $pwrp_exe. Filen ska namnges plc_'nodenamn'_'busnr'_'plcname'.opt, t ex plc_mynode_0999_plc.opt. Innehållet i filen skickas med som indata till länkaren, ld, och man lägger även in modulerna för plc-koden. I exemplet nedan antas att dessa moduler ligger i arkivet $pwrp_lib/libpwrp.a.

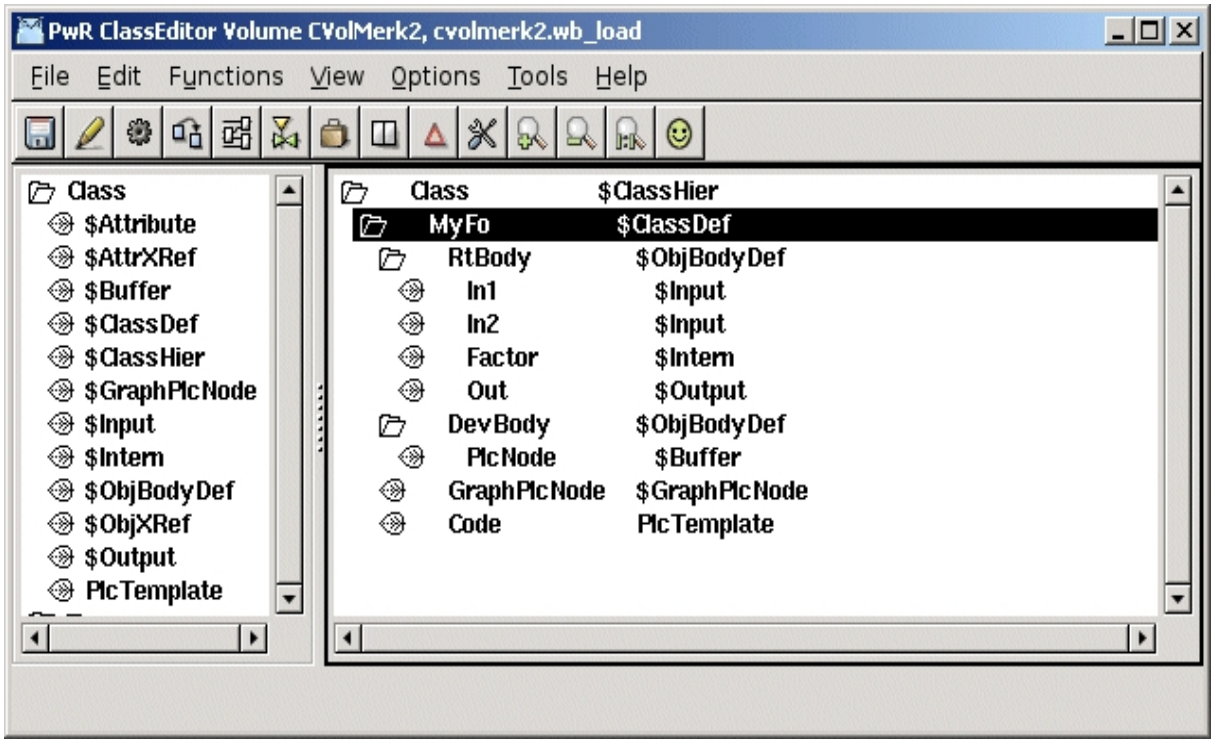
```
$pwr_obj/rt_io_user.o -lpwrp -lpwr_rt -lpwr_usbio_dummy -lpwr_usb_dummy -lpwr_pnak_dum
 -lpwr_cifx_dummy -lpwr_nodave_dummy -lpwr_epl_dummy
```

24.4.3.2 Funktions-objekt med plc-kod

Ett funktions-objekt där koden skrivs in form av plc-kod i plc-editor, definieras på liknande sätt som funktions-objeketet med c-kod ovan.

Funktionsobjekt klassen defineras med ett $ClassDef objekt under 'Class' objektet. Namnge objektet och aktivera Configure-Fo från popup-menyn för objektet. Nu skapas, förutom de objekt som även skapas för c-kods funktionsobjektet, ett Code objekt av klassen PlcTemplate. Det här objektet kan öppnas med plc-editorn, och här defineras koden för klassen.

Ingångar, interna attribute och utgångar i funktionsobjektet definieras, på samma sätt som för funktionsobjektet med c-kod, med $Input, $Intern och $Output attribut.



Definition av funktionsobjekt med plc-kod.

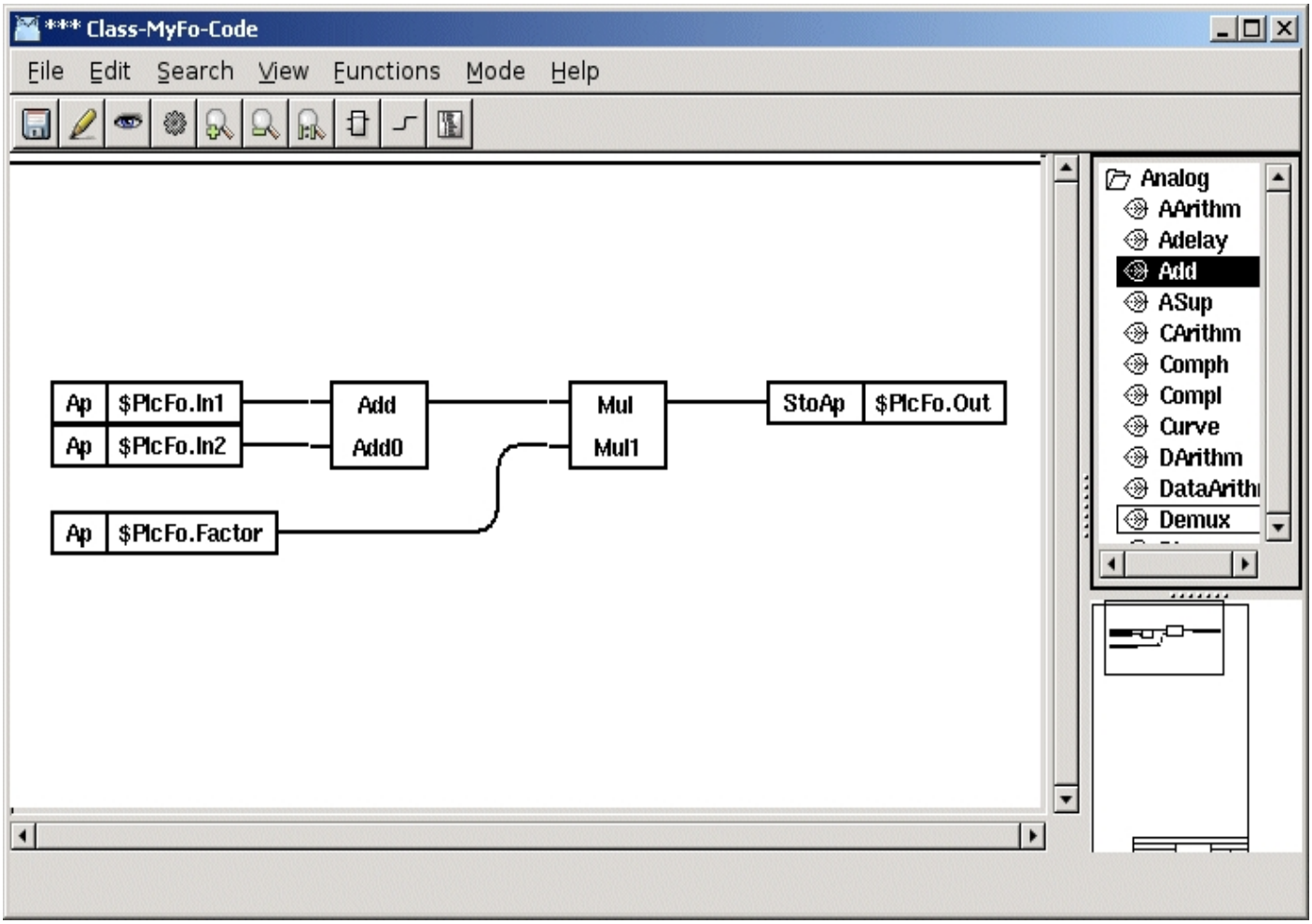
Kod

Genom att aktivera 'Open Program...' i popupmeny för Code objektet öppas plc-editorn. Här kan man nu skriva koden med funktions-objekts programmering. Koden skapas på samma sätt som ett vanligt program. Det som tillkommer är att hämta upp värden från ingångar och interna attribut, och att tilldela värden på utgångar.

Värden på ingångar, interna attribut, och även utgångar, hämtas upp i koden med GetDp, GetIp, GetAp eller GetSp objekt. Objekten knyts till attribut i objektet genom att attributet väljs ut i Template objektet för klassen, och 'Connect' aktiveras för Get-objektet. En symbolisk

referens $PlcFo läggs nu in i Get objektet. Den kommer sedan att bytas ut mot en referens till respektive instans, när koden för instansen kompileras.

Beräknade värden lagras i utgångar eller interna attribut med StoDp, StoIp etc. Dessa kopplas till attribut på samma sätt ingångarna, genom att välja ut attributen i Template objektet och aktivera 'Connect'.



Exempel på plc-kod för ett funktionsobjekt

Template koden i Code objektet behöver inte kompileras eller byggas. När en instans kompileras första gången, kopieras koden från template programmet till instansen.

Om template-koden ändras kommer instansernas kod att uppdateras nästa gång de kompileras (volymen med instanserna måste uppdateras med UpdateClasses först).

24.4.4 I/O klasser

I/O objekt är de objekt som hanteras av I/O hanteringen i ProviewR. De kan indelas i Agent, Rack, Card och Channel objekt. Vid knytning av nya I/O system till ProviewR måste man oftast skapa ny klasser av typerna Agent, Rack och Card. I/O objekt skapas med ett $ClassDef objekt där man i Flags sätter biten IoAgent, IoRack eller IoCard.

En närmare beskrivning av hur man skapar I/O objekt finns i Guide to I/O System.

24.4.5 Komponenter

En komponent är ett objekt, eller ett antal objekt som hanterar en komponent i anläggningen, det kan vara en ventil, en motor, en frekvensomformare etc. Tanken bakom komponent-begreppet är att man genom att skapa ett objekt (eller ett antal objekt) får med allt som behövs för att styra komponenten: ett objekt som innehåller data och signaler, ett funktionsobjekt med kod för att styra komponenten, objektsbild för HMI, simulerings objekt, I/O objekt för att knyta mot bus-kommunikation mm.

En komponent kan bestå av följande delar
- ett huvudobjekt.
- ett funktionsobjekt.
- ett simuleringsobjekt.
- ett eller flera I/O bus objekt.
- objektsbild för huvudobjektet.
- objektsbild för simulerings objektet.
- grafisk symbol för huvudobjektet.

24.4.5.1 Huvudobjekt

Huvudobjektet innehåller alla data som behövs för att konfigurera och göra beräkningar. Objektet läggs i planthierarkin, som ett individuellt objekt eller som en del av ett aggregat.

Ofta använder man BaseComponent:Component som superklass till komponentobjekt och för då med ett antal attribut, som Description, Specification, DataSheet etc.

Alla in- och utgångsignaler som finns på komponenten ska finnas i huvudobjektet. Di, Ii, Ai, Do, Io, Ao eller Co objekt läggs som attributobjekt. När man skapar instanser av komponenten måste signalerna kopplas till kanalobjekt. För t ex Profibus, kan man skapa ett modulobjekt som innehåller kanalerna, och, i klasseditorn, förkoppla signalerna i huvudobjektet till dessa kanaler. För varje instans behöver man då inte koppla varje signal för sig, utan kan göra en koppling mellan huvudobjekt och modulobjekt.

Speciella attribut

PlcConnect
Om det finns någon kod för objektet som ska exekveras av plc-programmet, skapar man ett funktionsobjekt för klassen. Detta måste kopplas till huvudobjektet, och denna koppling ska ligga i ett attribut med namnet 'PlcConnect' av typen pwrs:Type-$AttrRef.

SimConnect
Om det finns ett simulerings objekt så kopplas det till huvudobjektet med 'SimConnect' attribute av typen pwrs:Type-AttrRef.

IoConnect
Om det finns ett I/O-modul objekt kopplas detta med ett attribut, 'IoConnect' av typen pwrs:Type-AttrRef. Attributet hanteras av IoConnect metoden.

IoStatus
Om man vill hämta upp status från I/O-modul objektet skapar man attributet 'IoStatus' av typen pwrs:Type-$Status, och sätter Pointer biten i Flags.

Attributet kommer att tilldelas en pekare till I/O modulens Status attribut i runtime vid initieringen av I/O hanteringen. Status-attributet är av typen Status och kan t ex visas i en objektsbild med dynamiktypen StatusColor. Om man vill använda IoStatus i plc-koden för objektet, måste man tänka på att attributet är en pekare och hämta upp värdet upp med GetIpPtr.

SequenceReset
Det är möjligt att använda Grafcet sekvenser i en komponent. En skillnad från en ordinär sekvens är att resetobjektet ska vara definierat med ett attribut av klass Dv med namnet 'SequenceReset'i huvudobjektet. SubStep kan inte användas if sekvensen.

GraphConfiguration
GraphConfiguration är av typ Enum och används för att bestämma vilken objektsbild som ska öppnas för den aktuella instansen. Används av 'ConfigureComponent' metoden (se nedan).

DisableAttr

DisableAttr funktionen gör att möjligt att göra Disable på ett attribut i en instans. Om man har gjort Disable på ett attribut, visas attributet inte i navigatorn eller objekts editorn. Är attributet en signal, kommer den att ignoreras av I/O-hanteringen.

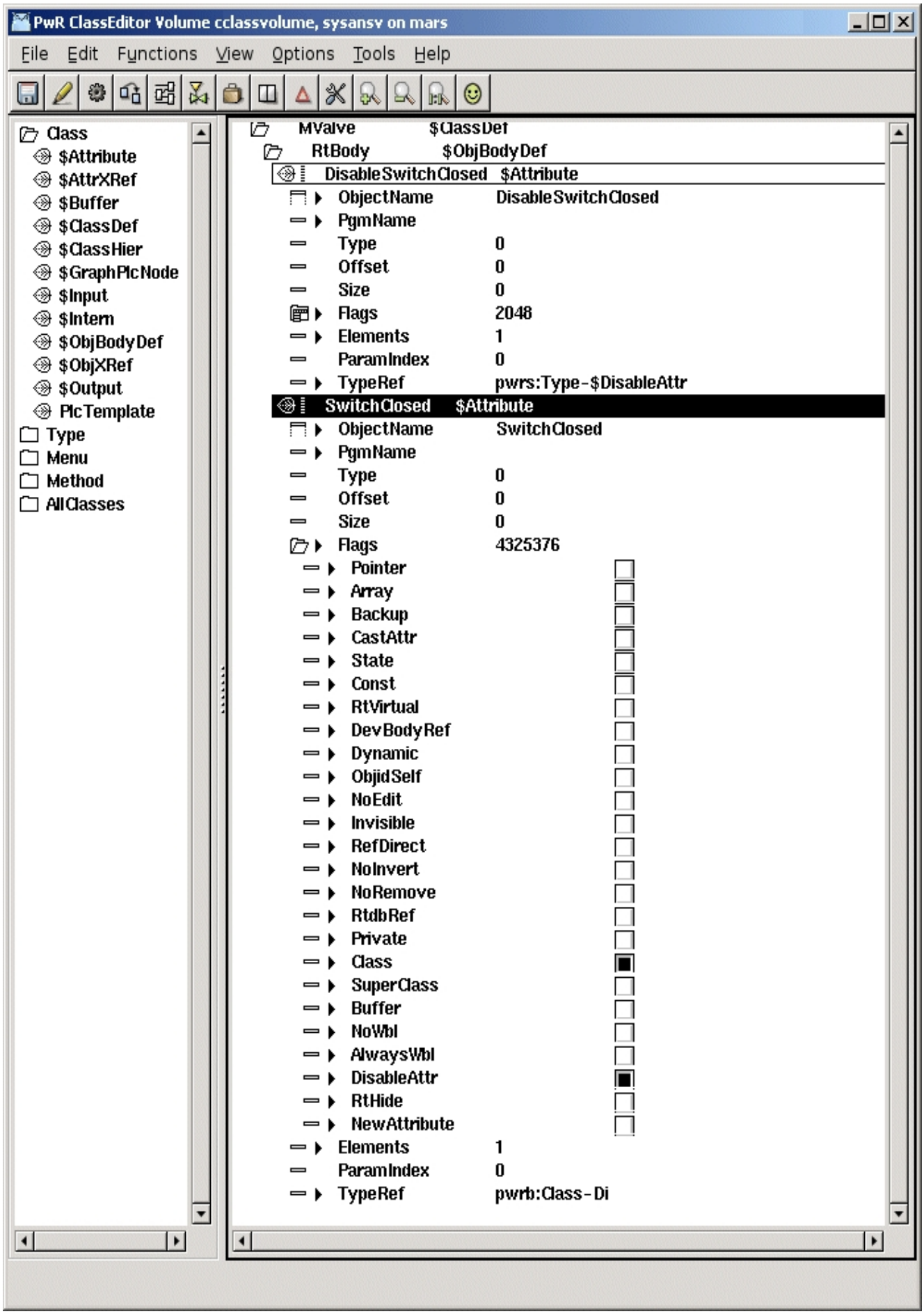
Funktionen används för komponenter som kan förekomma i olika varianter och konfigurationer. En magnetventil, till exempel, kan ha ett gränsläge som indikerar att ventilen är öppen, och ett som indikerar att ventilen är stängd. Totalt blir det fyra varianter av magnetventilen:

- inga gränslägen.
- gränsläge öppen.
- gränsläge stängd.
- både gränsläge öppen och gränsläge stängd.

Man skulle kunna göra fyra olika magnetventilsklasser, man problemen uppstår när man bygger aggregat av ventil objekten. Ett aggregat som innehåller ett ventilobjekt måste även det finnas i fyra olika varianter, och skulle aggregatet innehålla två ventilobjekt måste det finnas i 16 varianter. Genom att använda DisableAttr funktionen på gränslägesattributen kan man göra en magnetventilklass som täcker in alla fyra varianterna, och som även kan användas i aggregat-klasser.

DisableAttr för ett attribut läggs in på följande sätt.
- DisableAttr biten i Flags sätts för attributet.
- före attributet läggs ett attribut av typen pwrs:Type-$DisableAttr, med samma namn som attributet men med prefixet 'Disable'. Invisible biten i Flags ska sättas för DisableAttr attributet.

Exempel
I Magnetventilsklassen ovan representeras gränsläge stängd av attributet SwitchClosed som är en digital signal av type pwrb:Class-Di. Omedelbart före attributet läggs ett attribut med namnet 'DisableSwitchClosed' av typen pwrs:Type-$DisableAttr. För detta attribut sätts Invisible biten i Flags, och för SwitchClosed attributet sätts DisableAttr biten i Flags.


PwR ClassEditor Volume cclassvolume, sysansv on mars
File Edit Functions View Options Tools Help
Class
$Attribute
$AttrXRef
$Buffer
$ClassDef
$ClassHier
$GraphPlcNode
$Input
$Intern
$ObjBodyDef
$ObjXRef
$Output
PlcTemplate
Type
Menu
Method
AllClasses
MValve $ClassDef
RtBody $ObjBodyDef
DisableSwitchClosed $Attribute
ObjectName DisableSwitchClosed
PgmName
Type 0
Offset 0
Size 0
Flags 2048
Elements 1
ParamIndex 0
TypeRef pwrs:Type-$DisableAttr
SwitchClosed $Attribute
ObjectName SwitchClosed
PgmName
Type 0
Offset 0
Size 0
Flags 4325376
Pointer
Array
Backup
CastAttr
State
Const
RtVirtual
DevBodyRef
Dynamic
ObjidSelf
NoEdit
Invisible
RefDirect
NoInvert
NoRemove
RtdbRef
Private
Class
SuperClass
Buffer
NoWbl
AlwaysWbl
DisableAttr
RtHide
NewAttribute
Elements 1
ParamIndex 0
TypeRef pwrb:Class-Di

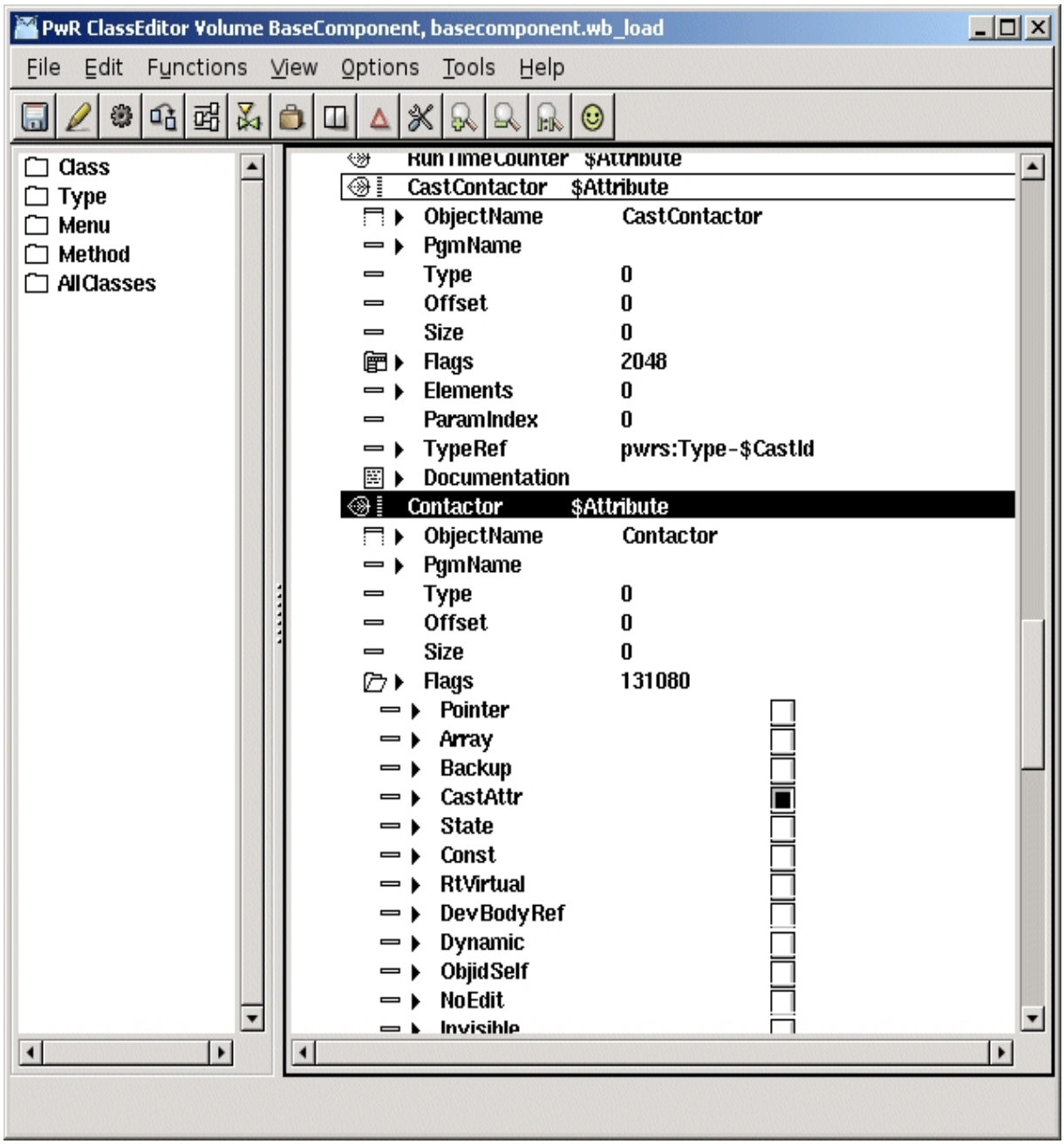
Attribut med disable funktion

Cast

Komponent-klasser byggs ofta relativt flexibla för att minimera antalet varianter. Ofta gör man en basklass som använder sig av DisableAttr funktionen för att kunna användas i ett antal olika konfigurationer. I exemplet ovan kan en klass för en magnetventil täcka in fyra olika konfigurationer av gränslägen genom att sätta DisableAttr på gränslägessignalerna. Man skapar även subklasser som är anpassade till en speciell ventil. En Durholt 100.103 till exempel har inga gränslägen, och man skapar då en subklass till magnetventilklassen, sätter Disable på båda gränslägena i template-objektet för subklassen, och lägger även in en länk till datablad och gör en del andra anpassingar. Resultatet blir en subklass som man kan använda för Durholt 100.103 ventiler utan att behöva konfigurera varje enskild instans.

Om vi nu bygger ett generellt aggregat som innehåller en magnetventil, och vill kunna utnyttja de subklasser som finns för magnetventilen, använder man Cast funktionen. Cast-funktionen innebär att ett attributobjekt kan castas som en subklass av dess ursprungliga klass, om subklassen har samma storlek. När ett attributobjekt castas hämtas defaultvärden, och därmed konfigureringar från subklassen. Även klassnamn och metoder hämtas från subklassen.

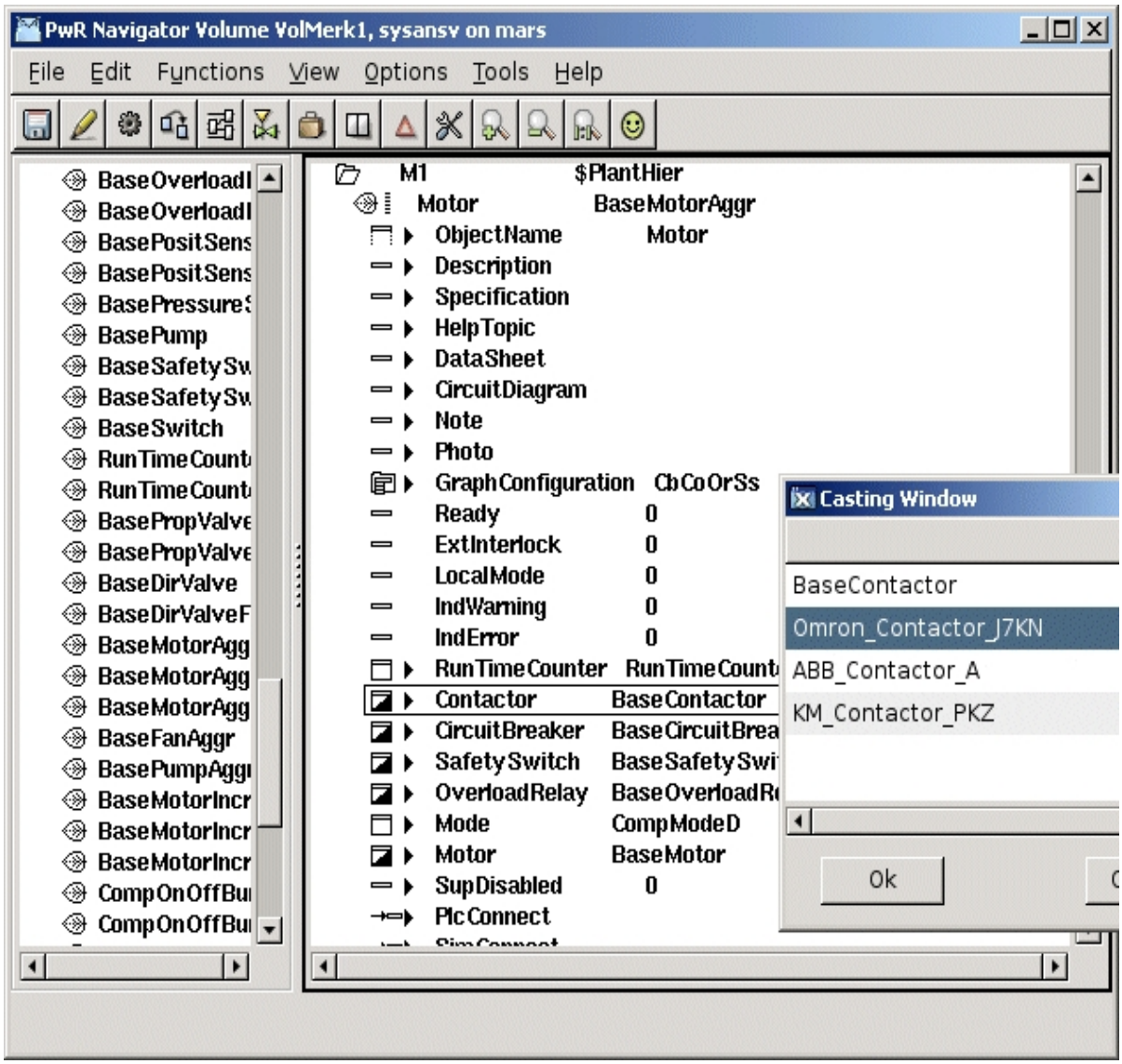
Cast funktionen för ett attribut läggs in på följande sätt:
- CastAttr biten sätt i Flags för attributet.
- Före attributet läggs ett attribut av typen pwrs:Type-$CastId med samma namn som attributet men med prefixet 'Cast'. Invisible biten i Flags ska sättas för cast attributet.



Kontaktor med cast attribut

Castningen av en instans utförs genom att aktivera 'Cast' metoden i popupmenyn för attributet. En lista med basklassen och samtliga subklasser visas, där man kan välja den klass attributobjektet ska castas som.

Om ett attribut har både cast och disable attribut ska castattributet placeras före disable attributet.



Castning av en instans

Metoder

Metoden ConfigureComponent

Ofta finns det många varianter på en komponent. I exemplet med magnetventilen ovan, hittade vi fyra olika varianter beroende på konfigurationen av gränslägen. För att förenkla konfigureringen av komponenten för användaren, kan man definiera metoden 'ConfigureComponent'.

ConfigureComponent metoden gör det möjligt att utifrån ett menyalternativ i popupmenyn sätta Disable på ett eller ett antal attribut, samt att välja en objektbild som är anpassad för den aktuella konfigurationen.

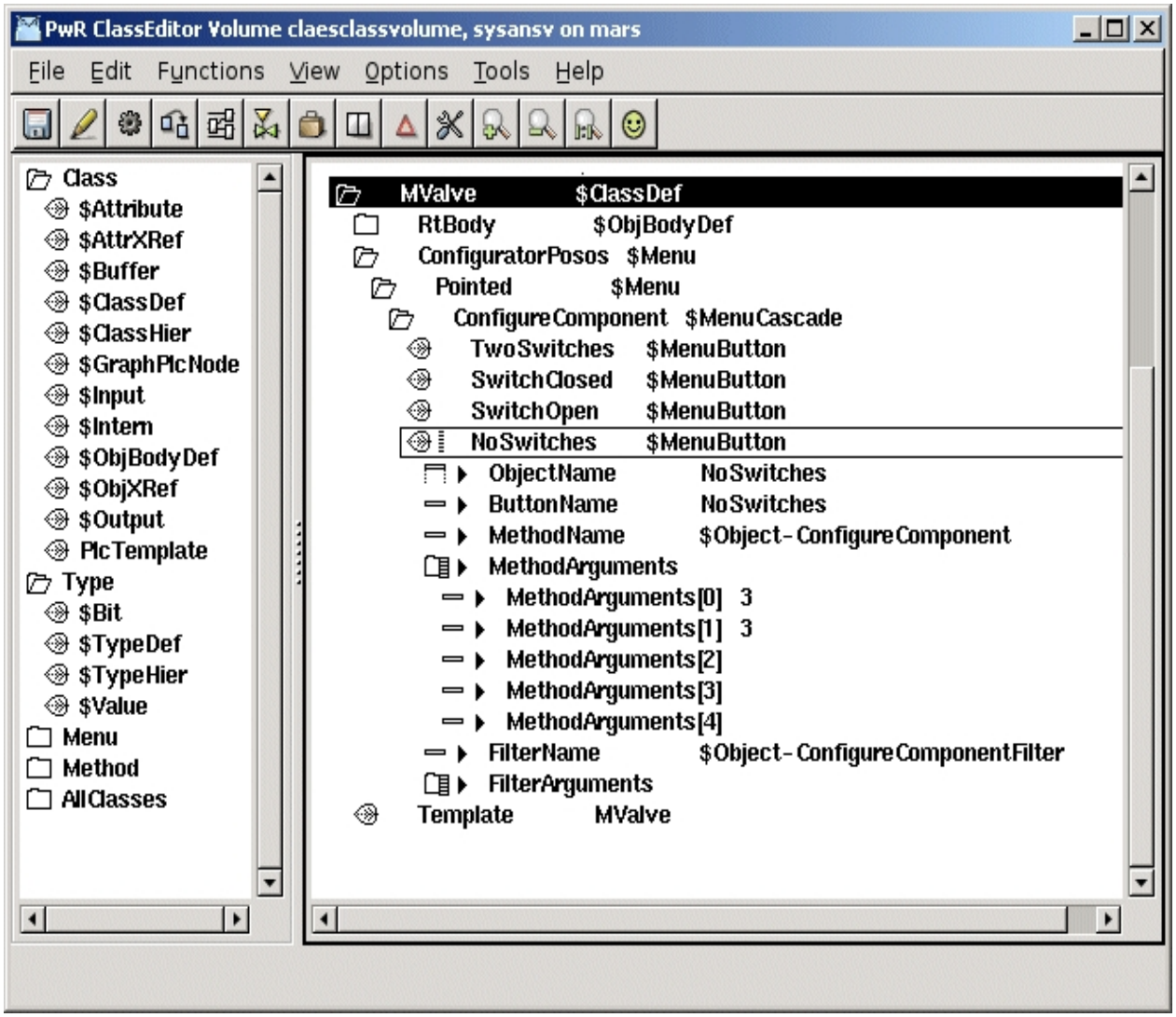
Meny

Menyalternativen för ConfigureComponent defineras med meny-objekt. Under $ClassDef objektet läggs ett $Menu objekt med namnet 'ConfiguratorPosos', vilket medför att menyn är synlig i editeringsmod när objektet är utpekat och utvalt. Under detta läggs ytterligare ett $Menu

objekt med namnet 'Pointed', och under detta ett $MenuCascade objekt med namnet 'ConfigureComponent'. Attributet ButtonName sätts till 'ConfigureComponent' för detta objekt. Under detta läggs slutligen ett $MenuButton objekt för varje konfigureringsalternativ. Namnet sätt lämpligen till namnet på konfigureringsalternativet, och läggs också in i attributet ButtonName. I attributet MethodName anges '$Object-ConfigureComponent' och i attributet FilterName '$Object-ConfigureComponentFilter'. Man ska även fylla i argument till metoden i MethodArguments. MethodArguments[0] innehåller en bitmask som avgör vilka attribut som ska göras Disable på i det aktuella menualternativet. Varje attribut som det är möjligt att göra Disable på representeras av en bit, och bitordningen motsvarar attributens ordning i objektet. MethodArguments[1] innehåller den grafiska representationen, se nedan.

Om vi tittar på magnetventilen, har den två attribut som kan göras Disable på, SwitchClosed och SwitchOpen. I bitmasken i MethodArguments[0] motsvarar SwitchClosed första biten, och SwichOpen andra biten, dvs om första biten är satt, görs Disable på SwitchClosed, och om andra biten är satt görs Disable på SwitchOpen. De fyra konfigureringsalternativen TwoSwitches, SwitchClosed, SwichOpen och NoSwitches motsvaras av följande masker

```
TwoSwitches   0   (både SwitchOpen och SwitchClosed är närvarande)
SwitchClosed  2   (Disable på SwitchOpen)
SwitchOpen    1   (Disable på SwitchClosed)
NoSwitches    3   (Disable på både SwitchOpen och SwitchClosed)
```



Konfigurering av komponent-attribut

Om man gör disable på ett attribut som är en komponent som i sin tur innehåller signaler, måste man göra disable även på signalerna i komponenten. I/O hanteringen tittar enbart på om den individuella signalen är satt Disable, och letar inte uppåt i attribut-nivåerna. För att göra Disable på en signal i ett komponent-attribut adderar man ett kommatecken samt namnet på komponenten följt av den Disable mask som gäller för komponenten i MethodArguments[0]. I ett objekt där man till exempel har gjort Disable på komponenterna Contactor och CircuitBreaker kan MethodArguments[0] se ut på följande sätt

```
3, Contactor 1, CircuitBreaker 1
```

där '3' är objektets Disable mask (som gör Disable på attributen Contactor och CircuitBreaker), och 'Contactor 1' medför att Disable sätts på ett signalattribut i Contactor, och 'CircuitBreaker 1' gör Disable på en signal i CircuitBreaker.

Det finns även en annan syntax med parenteser som tillåter mer än två nivåer. I det här exemplet är objektet ovan, Motor, del av ett större aggregat.

```
(5 (Motor 3 (Contactor 1, CircuitBreaker 1), Temp 1))
```

Komponent-attribut med individuell konfigurering

När ConfigureComponent metoden aktiveras tas Disable bort i alla komponentattribut för att återställa den tidigare konfigureringen. Ibland har man komponenter-attribut som inte ingår i objektets konfigurering, utan som ska konfigureras individuellt. Dessa komponenter ska inte återställas av ConfigureComponent metoden, och måste anges i MethodArguments[2], separerade med komma-tecken. I följande exempel ska komponent-attributen Motor och Contactor konfigureras med sina egna ConfigureComponent metoder och inte påverkas av objektets ConfigureComponent metod. I MethodArguments[2] skrivs

```
Motor, Contactor
```

Objektbild

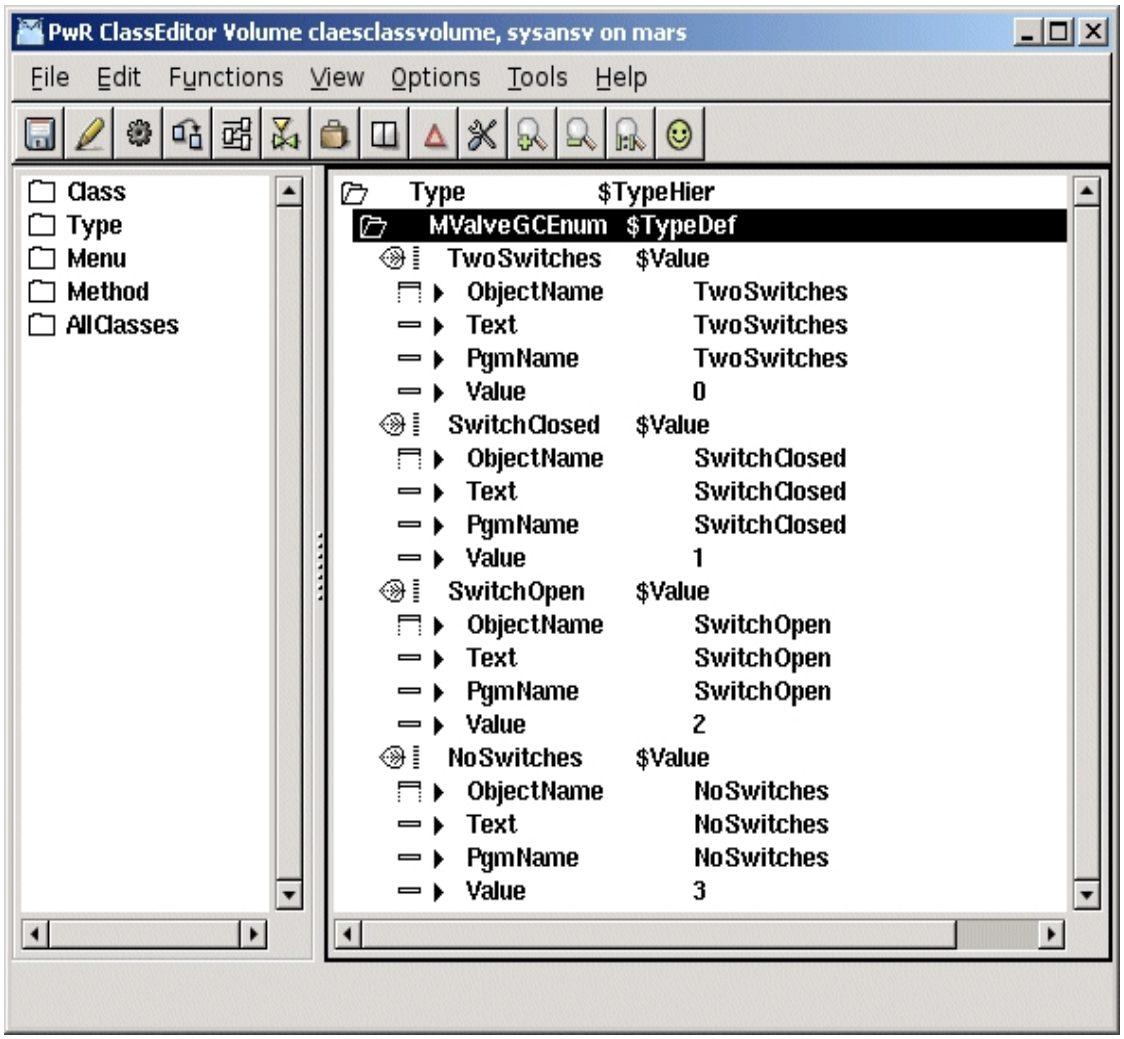
När man ritar objektsbilden för komponenten måste man ta hänsyn till vilken konfigurering som gäller. Om skillnaderna är små kan man använda Invisible dynamiken. Om skillnaden är större kan det vara smidigare att rita separata bilder för olika konfigurationer. Man lägger då in ett attribut i objektet med namnet GraphConfiguration av typen Enum. Lämpligen skapar man en speciell enum-typ med aktuella alternativ. Om GraphConfiguration är 0 används standard grafen, annars sätts de numeriska värdet i GraphConfiguration som suffix på graf-namnet.

För exemplet med magnetventilen, MValve, gör vi en enum-typ, MValveGCEnum, och definerar värdena

```
TwoSwitches  0
SwitchClosed 1
SwitchOpen   2
NoSwitches   3
```

För konfigurationen TwoSwitches, med värdet 0, ritar vi en objektsgraf med namnet mvalve.pwg. För SwitchClosed, med värdet 1, namnger vi grafen mvalve1.pwg, för SwitchOpen mvalve2.pwg och för NoSwitches mvalve3.pwg.

Vi lägger även in enum-värdet i MethodArguments[1] i $MenuButton objektet för den aktuella konfigurationen. Det kommer att medföra att GraphConfiguration sätts till detta värde när detta menyalternativ aktiveras.



Enumeration typ för GraphConfiguration

24.4.5.2 Funktionsobjekt

Funktionsobjektet är gränssnittet i plc-programmet. Det definierar in och utgångar som kan kopplas till andra funktionsobjekt i plc-editorn. Till skillnad från ett vanligt funktionsobjekt jobbar koden i funktionsobjektet för en komponent även mot data i huvudobjektet.

Man kan skriva koden i plc-kod eller i c-kod.

plc-kod

Om man vill att koden i funktionsobjektet ska vara synlig, och det finns behov av att kunna gå in och köra PlcTrace i koden, är det lämpligt att använda ett funktionsobjekt med plc-kod.

Skapa ett $ClassDef objekt och sätt namn på objektet. Lämpligen tar man samma namn som huvudklassen föjt av suffixet 'Fo', t ex MyComponentFo. Aktivera därefter Configure-ConnectedFo i popupmenyn.

Under RtBody har det skapats ett PlcConnect attribut av typen AttrRef, som kommer att innehålla länken till huvudobjektet när man har gjort connect i plc-editorn.

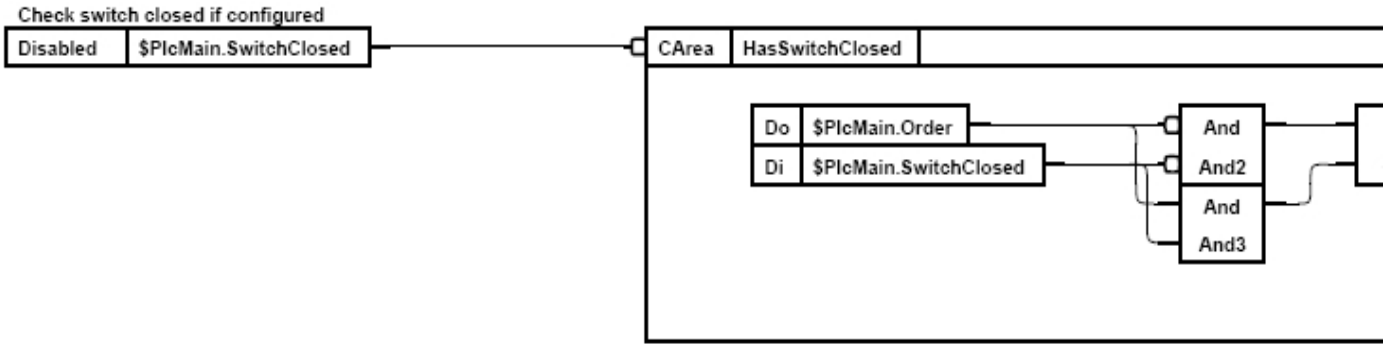
Konfigurera in och utgångar med $Input och $Output objekt under RtBody objektet. Man kan även lägga in $Intern objekt, men vanligtvis lagrar man denna typ av data i huvudobjektet. Observera

att ordningen bland attributobjekten ska var $Input, $Intern, $Output.

Koden skapas genom att öppna plceditorn för Code objektet. I koden hämtar man upp värden från ingångar genom att välja ut ingångsattributen i template objekete för funktionsobjektet i navigatören och aktivera connect funktionen. Utgångar sätts på motsvarande sätt. När data ska hämtas eller sättas i huvud-objektet, väljar man ut attribut i huvudobjektets template-objekt. Referenser till funktionsobjektet visas i plc-koden med symbolen $PlcFo och referenser till huvudobjektet med symbolen $PlcMain.

Om objektet innehåller komponenter läggs funktionsobjekten för dessa komponenter in i plc-koden.

Har man DisableAttr på signaler eller andra attribut måste man hantera detta med villkorlig exekvering i koden. En signal som är satt Disable för inte sättas eller läsas i koden. Man använder objektet Disabled under mappen Other, för att hämta upp om ett attribut är satt Disable eller inte. Detta kan sedan kopplas till ett CArea som står för den villkorliga exekveringen.



Villkorlig exekvering med Disabled och CArea

c-kod

Ett funktionsobjekt med c-kod konfigureras med ett $ClassDef objekt. Sätt namn på objektet och aktivera därefter Configure-ConnectedCCodeFo i popupmenyn.

Under RtBody har det skapats två attribut, PlcConnect av typen AttrRef och PlcConnectP som är en pekare. I PlcConnect läggs referensen till huvudobjektet in, när man gör connect på en instans i plc-editorn. Vid initieringen av plc-programmet i runtime hämtar man, med hjälp av referensen, en pekare till huvudobjektet. Pekaren läggs i PlcConnectP. Detta görs i c-koden som är uppdelad i en init funktion som exekveras vid initieringen av plc-programmet, och en exec function som exekveras varje scan. För funktionsobjektet MyComponentFo med ingångarna In1 och In2 och utgången Out2 kan koden se ut på följande sätt.

```
void MyComponentFo_init( pwr_sClass_MyComponentFo  *o)
{
  pwr_tDlid dlid;
  pwr_tStatus sts;

  sts = gdh_DLRefObjectInfoAttrref( &o->PlcConnect, (void **)&o->PlcConnectP, &dlid);
  if ( EVEN(sts))
    o->PlcConnectP = 0;
}

void MyComponentFo_exec( plc_sThread      *tp,
                         pwr_sClass_MyComponentFo *o)
{
```

```
  pwr_sClass_MyComponent *co = (pwr_sClass_MyComponent *) o->PlcConnectP;

  if ( !co)
    return;

  o->Out = co->Value = co->Factor * (*o->In1P + *o->In2P);
}
```

24.4.5.3 Simuleringsobjekt

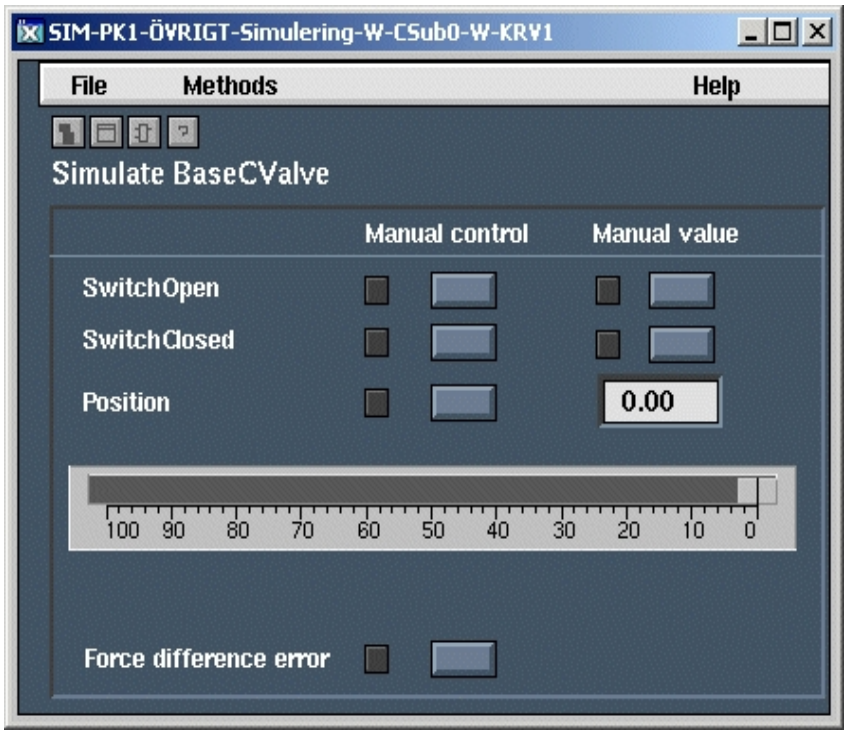
Ett simulerings objekt används för att simulera processen, både vid normala förhållanden och när olika felfall uppträder. Simuleringsobjektet läser av utgångs-signaler (Do, Ao, Io) i huvudobjektet och sätter värden på ingångs-signaler (Di, Ai, Ii, Co). Objektet är ett funktionsobjekt som kan innehålla in och utgångs attribut, men det vanliga är att dessa saknas och att objektet jobbar mot data i huvudobjektet och mot interna attribut som konfigurerar simuleringen och utlöser olika felfall. Simuleringsobjektet har ofta en objektsgraf som i runtime kan öppnas med huvudobjektets Simulate metod.

Ett simuleringsobjekt kopplas till huvudobjektet med en connectmetod på samma sätt som ett vanligt funktionsobjekt. Men simuleringsklassen har en annan connect-metod än Fo klassen. Huvudobjektet ska innehålla ett attribut 'SimConnect' av typen pwrs:Type-$AttrRef, i vilket connectmetoden lägger in objektsidentiteten för simuleringsobjektet när huvudobjekt och simulerings kopplas ihop.

En simulerings klass skapas på samma sätt som en funktionsobjekts klass, och kan skrivas antingen i c- eller plc-kod. Klassen brukar namnges med samma namn som huvudklassen, följt av suffixet 'Sim'.

Skapa ett $ClassDef objekt och sätt namn på objektet. Aktivera Configure-ConnectedFo eller Configure-ConnectedCCodeFo. Lägg in eventuella $Input, $Intern och $Output attribut, och skriv koden som plc eller c-kod. Ändra connectmethod i GraphPlcNode till 26.

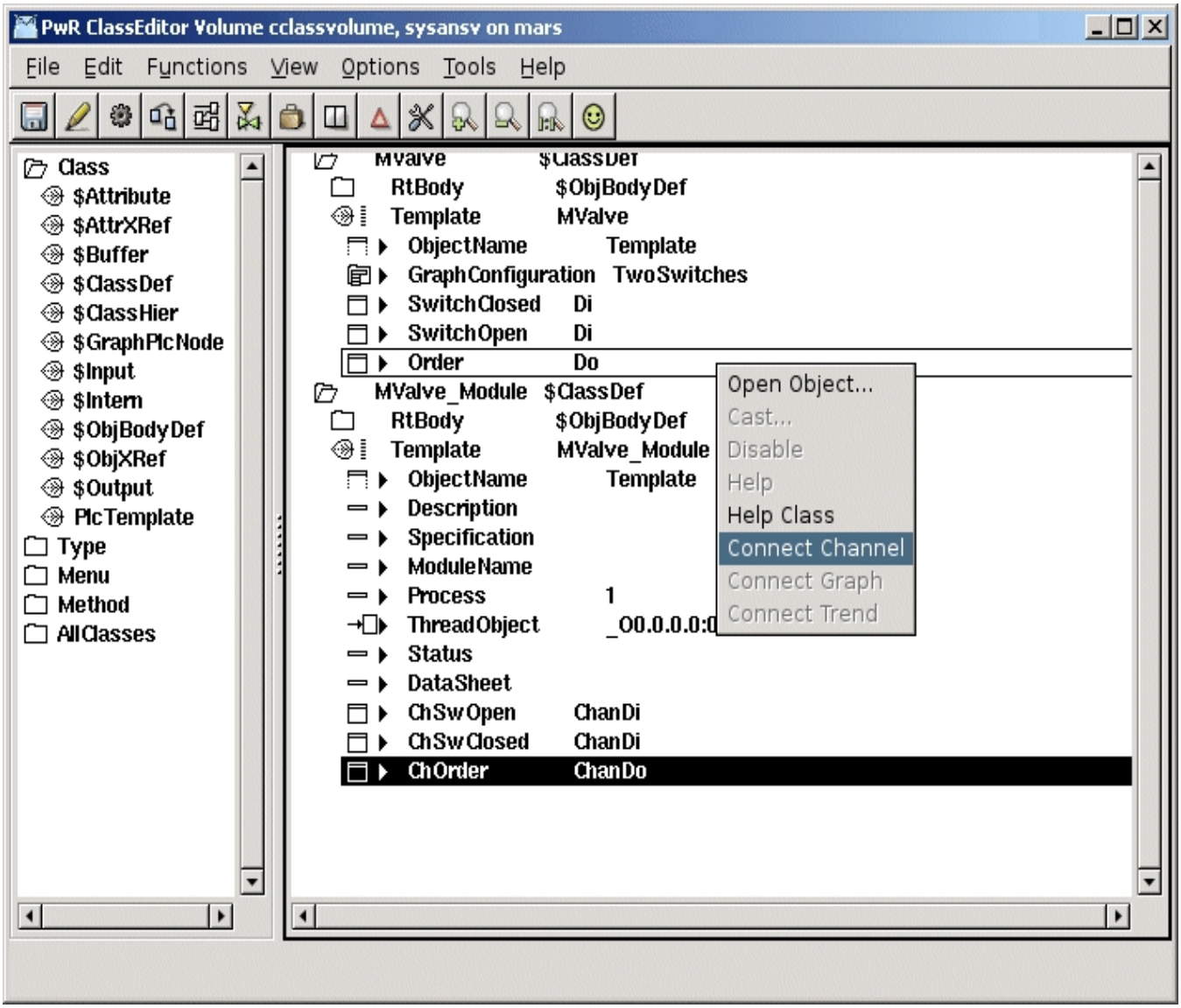
Objektsbilden för simulerings objekt ritas ofta med mörkblå bakgrundsfärg och vit text för att enkelt kunna skiljas från andra objektsbilder. Notera att attribut i huvudobjektet kan refereras från bilden med referensnotationen '&', t ex &($object.PlcConnect).IndError##Boolean.



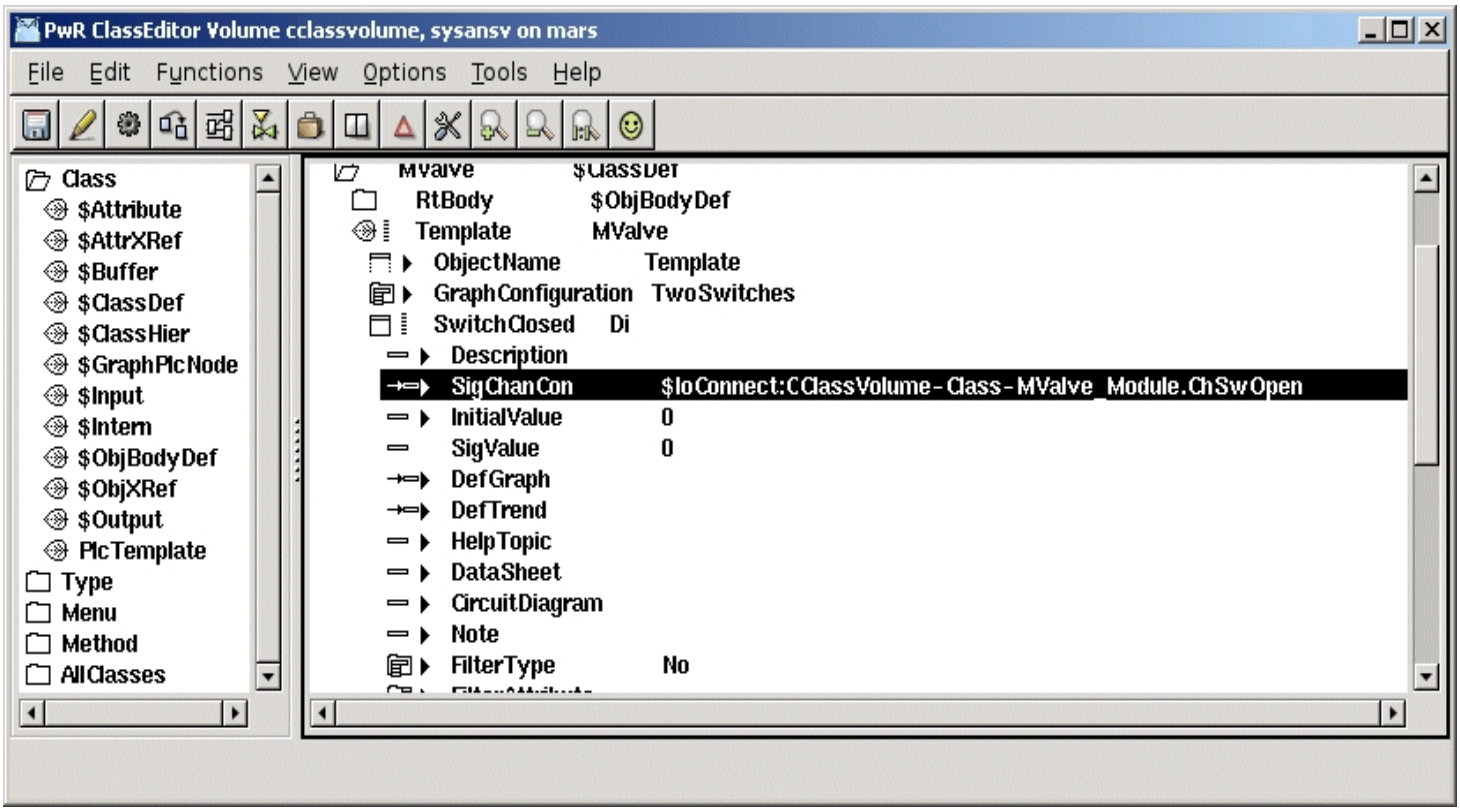
Objektsbild för ett simuleringsobjekt

24.4.5.4 I/O-modul objekt

Ett huvudobjekt för en komponent kan innehålla signalobjekt av typen Ai, Di, Ii, Ao, Do, Io och Co. Signalerna i en instans måste kopplas till kanalobjekt i nodehierarkin. I t ex profibus kan man skapa modulobjekt, där kanalerna är anpassade till den dataarea som tas emot resp skickas på bussen. I det fall signalerna i en komponent, omfattas av ett modulobjekt, kan man lägga in symboliska referenser till kanalobjekten i signalobjekten, vilket medför att man endast behöver koppla komponenten med modulobjektet, inte varje enskild signal för sig. De symboliska referenserna läggs in i komponentens template-objekt genom att man gör connect mellan signalerna i template-objektet och kanalerna i I/O modulklassens template objekt. De symboliska referenserna är av typen $IoConnect konverteras till verkliga referenser vid initieringen av I/O hanteringen i runtime.



Signal kopplas med kanal i I/O modul



Symbolisk referens till kanalobjekt

24.4.5.5 Objektsbild

Objektsbilder har samma namn som komponenten, men med små bokstäver. För klasser som tillhör ProviewRs bassystemet adderar man prefixed 'pwr_c_'. Bilderna editeras på vanligt sätt i Ge. I dynamiken ersätts objektsnamnet med '$object'. Objektsbilder för bassystemet ritas med följande riktlinjer.

Meny
Det ska finnas en meny med pulldown menyerna File, Methods, Signals och Help.

File ska ha meny följande entryn

```
Print   Command     print graph/class/inst=$object
Close   CloseGraph
```

Methods ska ha följande entryn

```
Help            Invisible   $cmd(check method/method="Help"/object=$object)
                Command     call method/method="Help"/object=$object
Note            Invisible   $cmd(check method/method="Note"/object=$object)
                Command     call method/method="Note"/object=$object
Trend           Invisible   $cmd(check method/method="Trend"/object=$object)
                Command     call method/method="Trend"/object=$object
Fast            Invisible   $cmd(check method/method="Fast"/object=$object)
                Command     call method/method="Fast"/object=$object
Help            Invisible   $cmd(check method/method="Photo"/object=$object)
                Command     call method/method="Photo"/object=$object
DataSheet       Invisible   $cmd(check method/method="DataSheet"/object=$object)
                Command     call method/method="DataSheet"/object=$object
```

```
Hist Event...    Invisible   $cmd(check method/method="Hist Event..."/object=$object)
                 Command     call method/method="Hist Event..."/object=$object
Block Events...  Invisible   $cmd(check method/method="Block Events..."/object=$objec
                 Command     call method/method="Block Events..."/object=$object
RtNavigator      Invisible   $cmd(check method/method="RtNavigator"/object=$object)
                 Command     call method/method="RtNavigator"/object=$object
Open Object      Invisible   $cmd(check method/method="Open Object"/object=$object)
                 Command     call method/method="Open Object"/object=$object
Open Plc         Invisible   $cmd(check method/method="Open Plc"/object=$object)
                 Command     call method/method="Open Plc"/object=$object
Circuit Diagram  Invisible   $cmd(check method/method="Circuit Diagram"/object=$objec
                 Command     call method/method="Circuit Diagram"/object=$object
HelpClass        Invisible   $cmd(check method/method="HelpClass"/object=$object)
                 Command     call method/method="HelpClass"/object=$object
```

Signals ska innehålla alla signaler i komponenten och öppna objektsbilden för respektive signal. Exempel

```
SwitchOpen Di    Command     open graph/class /inst=$object.SwitchOpen
SwitchClosed Di  Command     open graph/class /inst=$object.SwitchClosed
Order Do         Command     open graph/class /inst=$object.Order
```

Help menyn ska innehålla Help och HelpClass

```
Help             Command     call method/method="Help"/object=$object
HelpClass        Command     call method/method="HelpClass"/object=$object
```

Verkygspanel

Vertygspanelen innehåller knappar för olika metoder. Dynamiken är densamma som i Method menyn ovan. Längst till höger finns även en knapp för simuleringsobjektets objektsbild. Denna har dynamiken

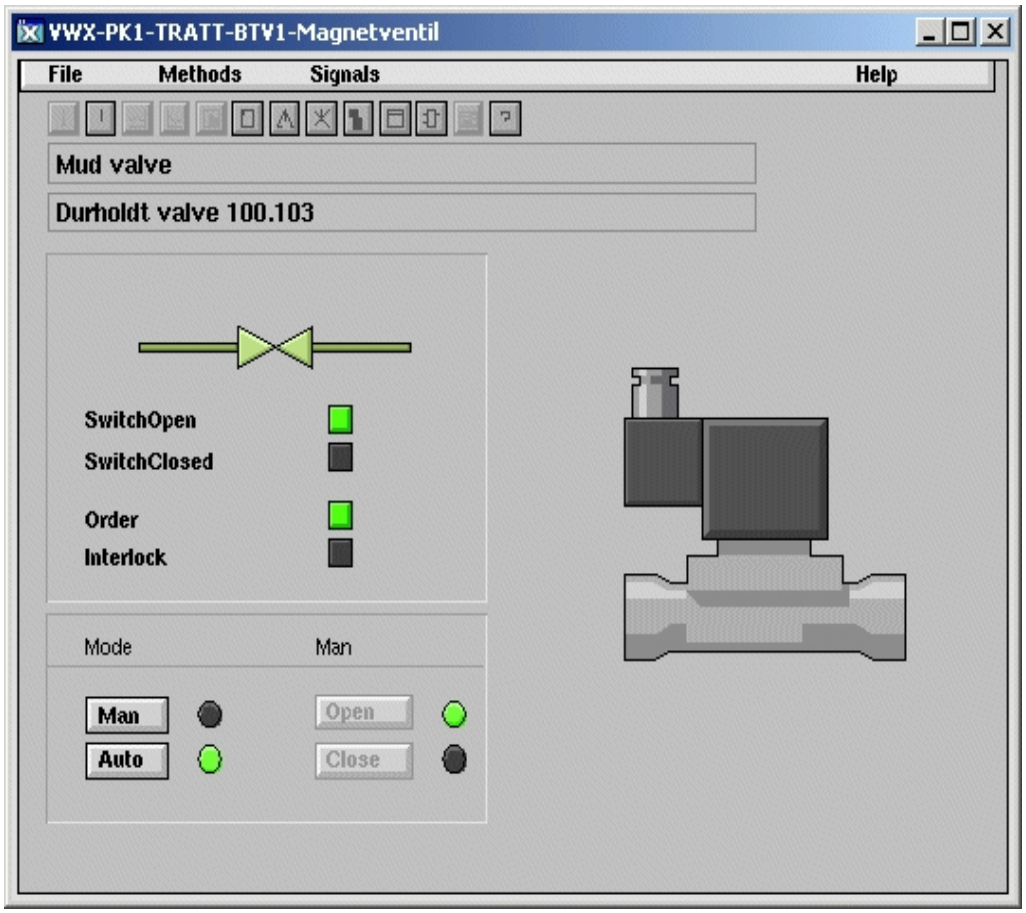
```
Invisible   $cmd(check method/method="Simulate"/object=$object)
Command     call method/method="Simulate"/object=$object
```

Under metodknapparena ligger två textfält som visar Descripton och Specification attributen i komponenten med följande dynamik

```
Description     Value.Attribute $object.Description##String80
                Value.Format    %s
Specification   Value.Attribute $object.Specification##String80
                Value.Format    %s
```

Längst ner i bilden visas eventuellt Notes meddelande med en knapp för att ändra eller ta bort meddelandet.

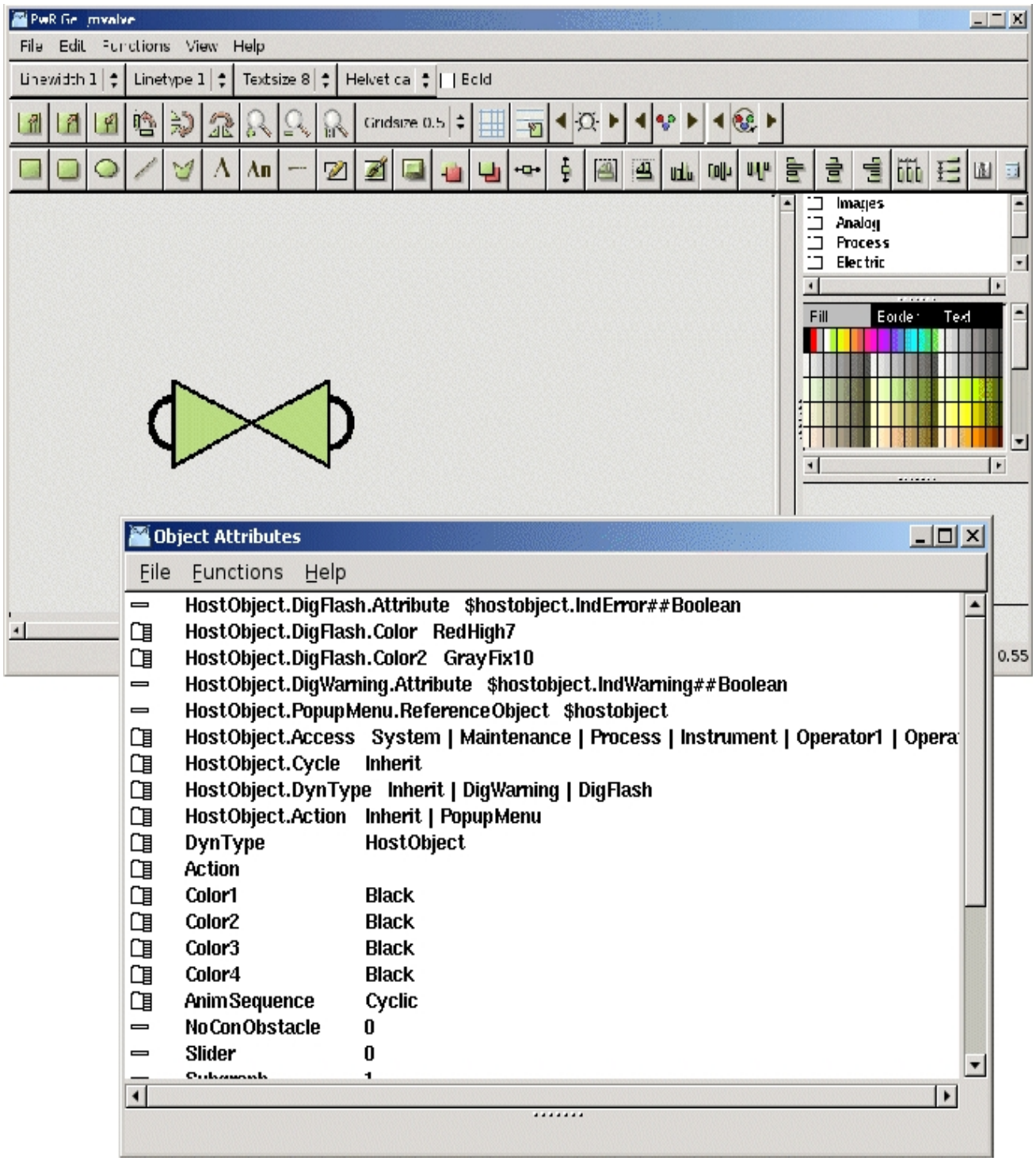
```
Notes knapp     Invisible       $object.Note##String80
                Command         call method/method="Note"/object=$object
Notes text      Value.Attribute $object.Note##String80
                Value.Format    %s
```



Objektbild

24.4.5.6 Grafisk symbol

Den grafiska symbolen ritas i Ge och ges dynamiken HostObject. För dynamiken i host objekt kopplas olika typer av dynamik till attribute i objektet. Objektsnamnet ersätts med '$hostobject' i dynamiken. Ofta skapar man attributen IndWarning och IndError i huvudobjektet och färgar symbolen gul resp röd, eller blinkande röd, om dessa är satta.



Grafisk symbolen ritas i Ge med HostObject dynamik

24.5 Bygga klassvolym

När man bygger klassvolymen skapas en laddatafil och två struct-filer.

Laddatafil
Laddatafilen läggs på $pwrp_load och har samma namn som volymen, men med små bokstäver. Filtypen är .dbs, dvs laddatafilen för volymen CVolMerk1 blir $pwrp_load/cvolmerk1.dbs.

Tiden när man skapar laddatafilen lagras i filen. Dessutom lagras versionen av andra klassvolymer som laddatafilen är beroende av. Vid uppstart av runtime, kontrolleras att de versioner som finns i systemet överensstämmer med de versioner som finns registrerade i laddatafilen. Om någon version inte stämmer, får man 'Version mismatch' och uppstarten avbryts.

Man kan se versionen av en laddatafil och versionen av andra volymer den är beroende av med wb_ldlist.

```
$ wb_ldlist $pwrp_load/cvolmerk1.dbs
Volume     CVolMerk1       21-DEC-2007 13:52:05.22 (1198241525,227130443) 25364
VolRef     CVolMerk1       21-DEC-2007 13:52:05.22 (1198241525,227130443) 25364
VolRef     pwrs            12-DEC-2007 08:35:06.98 (1197444906,983976467) 1
VolRef     pwrb            12-DEC-2007 08:35:09.93 (1197444909,930182284) 2
VolRef     BaseComponent   12-DEC-2007 08:35:26.92 (1197444926,926679004) 10
```

Structfiler
Vid byggandet av en klassvolym skapas två include-filer, en .h, och en .hpp.

Om klassvolymen innehåller funktionsobjekt, eller klasser som används i CArithm eller DataArithm objekt, måste man inkludera .h filen i $pwrp_inc/ra_plc_user.h.

Uppdatera klasser
När man byggt klassvolymen, måste man uppdatera klasserna i rot- eller subvolymer i projektet. Uppdateringen aktiveras i konfiguratören för respektive rot eller subvolym, från menyn med 'Functions/Update Classes'. Om en klass är ändrad, kommer instanser av klassen att anpassas till den förändrade klassen. Vidare kommer alla referenser till instanser av klassen att uppdateras.

24.6 Dokumentation av klasser

För $ClassDef och $Attribute objekt finns ett dokumentationsblock, som fylls i från objektseditorn. Från dokumentationsblocken och klassbeskrivningen genereras dokumentation för klasserna på xtthjälp och html format när klassvolymen byggs.

Dokumentationsblocket för $ClassDef objektet bör innehålla en beskrivning av klassens funktion och användningsområde, documentationsblocket för $Attribut en beskrivning av det aktuella attributets funktion.

24.6.1 Generering av Xtt hjälpfiler

Hjälpfiler för xtt genereras med kommandot

```
co_convert -xv -d $pwrp_exe/ $pwrp_db/userclasses.wb_load
```

Kommandot genererar en hjälpfil $pwrp_exe/'volymsnamn'_xtthelp.dat som man lämpligt-vis lägger en länk till i projektets xtt-hjälpfil $pwrp_exe/xtt_help.dat:

Exempel för klassvolymen cvolvhxtank:

<topic> index

...

Användar klasser<link>cvoltank,"",$pwrp_exe/cvoltank_xtthelp.dat

</topic>

...

<include> $pwrp_exe/cvoltank_xtthelp.dat

24.6.2 Generering av html dokumentation

html-filer genereras med kommandot

```
co_convert -wv -d $pwrp_web/ $pwrp_db/userclasses.wb_load
```

Kommandot genererar bl a filen $pwrp_web/'volymsnamn'_index.html som innehåller startsidan för klass-dokumentationen. Denna tillsammans med övriga filer ($pwrp_web/'volymsnamn'_*.html) kopieras till lämplig filkatalog på web-servern.

En länk till dokumentationen skapar man lämpligen med ett WebLink-objekt med URL'en 'volymsnamn'_index.html

Vill man kunna visa c-structarna för klasserna, konverterar man h-filen med co_convert

```
co_convert -cv $pwrp_web/ $pwrp_inc/'volymsnamn'classes.h
```

Vill man dessutom visa koden för plc-objekt ska man lägga in aref-taggar i c eller h-filen för koden och konvertera den med

```
co_convert -sv -d $pwrp_web/ 'filnamn'
```

24.6.3 ClassDef

Exempel

```
@Author Homer Simpson
@Version 1.0
@Code ra_plc_user.c
@Summary Brief description of this class
Description of
this class.

See also
@link Example plat.html
@classlink AnotherPlate cvolvhxn2r_anotherplate.html
```

Taggar

Author	Redaktör
Version	Version av klassen
Code	Fil som innehåller kod för klassen

Summary	Sammanfattning
Link	Godtycklig länk
Classlink	Länk till annan klass
wb_load syntax	

24.6.3.1 @Author

Författare. Kan utelämnas.

Syntax
```
@Author 'name of author'
```

24.6.3.2 @Version

Version. Kan utelämnas.

Syntax
```
@Version 'version number'
```

24.6.3.3 @Code

För klasser med plc-kod kan man ange namnet på c-filen. Kan utelämnas.
Även c-filen måste konverteras med: co_convert -c -d $pwrp_web/ 'filnamn'

Syntax
```
@Code 'filename'
```

24.6.3.4 @Summary

Kort beskrivning på en rad.
Denna visas i indexfilen i xtt-hjälpfilen. Används ej i html.
Kan utelämnas.

Syntax
```
@Summary 'text'
```

24.6.3.5 @Link

Här kan läggas in en länk till godtycklig URL. Kommer enbart att visas i html-dokumentationen, ej i xtt. Länken måste ligga efter beskrivningen av klassen.

Syntax
```
@Link 'URL'
```

24.6.3.6 @Classlink

Här kan läggas in en länk till en annan klass. Denna länk fungerar både i html och xtt.
Länken måste ligga efter beskrivningen av klassen.

Syntax
```
@Classlink 'html-filename'
```

24.6.3.7 wb_load syntax

Info om en klass skrivs före $ClassDef raden.

```
!
```

```
!/**
!  @Author Homer Simpson
!  @Version 1.0
!  @Code ra_plc_user.c
!  @Summary Brief description of this class
!  Description of
!  this class.
!
!  See also
!  @link Example plat.html
!  @classlink AnotherPlate cvolvhxn2r_anotherplate.html
!*/
!
Object        Plat    $ClassDef 1
```

!/**

Start av ett dokumentations block.
All text mellan !/** och !*/ kommer att skrivas ut i oförvanskad form som beskrivning för klassen.

!*/

Avslutning av ett dokumentations block

24.6.4 Attribute

Exempel

```
@Summary Plåtens längd
En grundligare beskrivning
av attributet Langd...
```

@Summary

Kort beskrivning på en rad. Om det finns en @Summary läggs denna text in i html-filens tabell över attribut. Om den inte finns läggs beskrivningen hela beskrivningen in.
Används ej i xtt.

wb_load syntax

24.6.4.1 wb_load syntax

Info om ett attribute skrivs före $Attribute, $Input, $Output eller $Inter raden.

```
!/**
!  @Summary Plåtens längd
!  En grundligare beskrivning
!  av attributet Langd...
!*/
Object    Langd $Attribute 3
  Body             SysBody
    Attr           TypeRef = "pwrs:Type-$Float32"
  EndBody
```

```
      EndObject
```

!/**

Start av ett dokumentations block.
All text mellan !/** och !*/ kommer att skrivas ut i oförvanskad form som beskrivning för attributet, förutom nedanstående taggar.

!*/

Avslutning av ett dokumentations block

24.6.5 Syntax för c- och h-filer

Om man vill lägga in länkar till c- och h-filer måste även dessa konverteras till html. Det finns även en funktion att lägga in bokmärken. ¨

Struct-filen för klasserna genereras automatiskt med bokmärken.

```
/**
   MyPlcClass

   Description for the class that is not displayed anywhere but in the code.

   @aref MyPlcClass MyPlcClass
*/
void MyPlcClass_exec(...)
```

@aref

@aref måste ligga inom ett /*_* ... */ block. Inom blocket kan även finnas kommentarer som inte hanteras vid konverteringen.

Syntax

```
@aref 'bookmark' 'text'
```

25 Administration

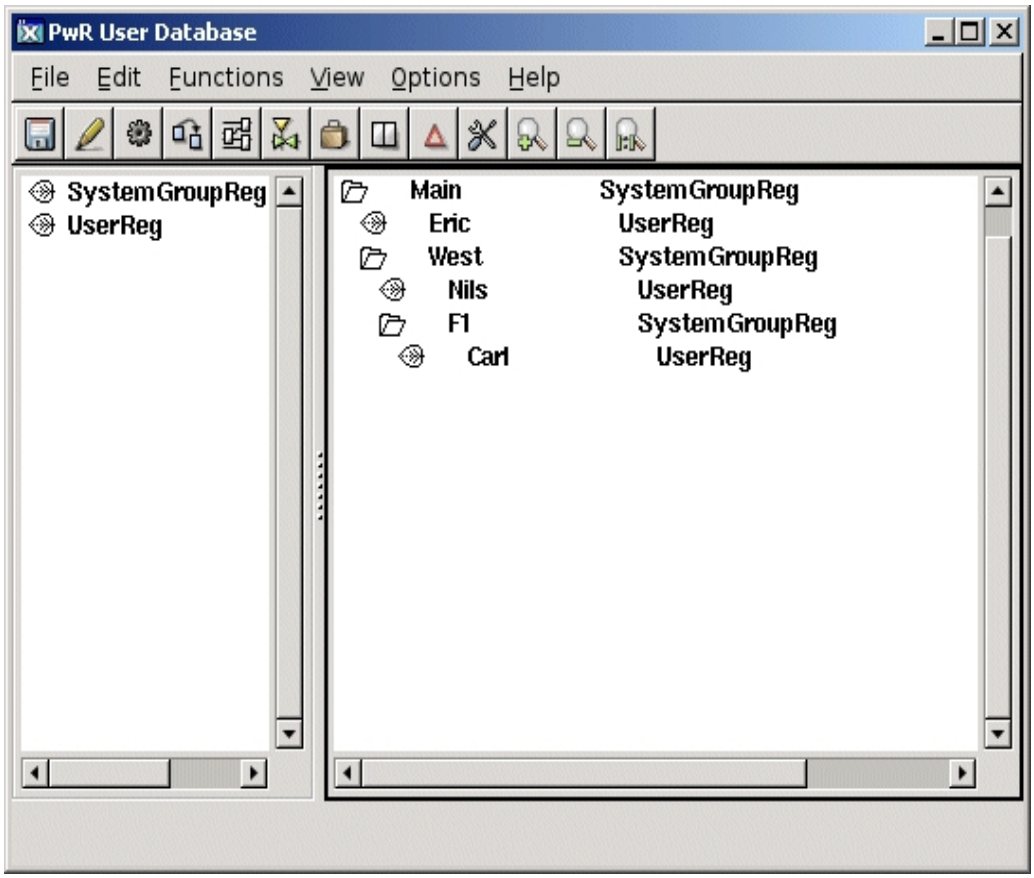
25.1 Användare

För att få tillgång till ProviewR's utvecklings och runtime miljöer måste man logga in med användarnamn och passerord. Användare konfigureras i en användardatabas och tilldelas privilegier som bestämmer användarens behörighet att göra ändringar i systemet.

System som delar samma användare är grupperade i en systemgrupp, och utgående från systemgruppen definieras användarna. Man kan bygga en hierarki av systemgrupper där barngrupper ärver användare av sin förälder, och ytterligare användare kan definieras för varje barn.

Ett system kopplas till en systemgrupp med attributet SystemGroup i $System objektet. Notationen för en systemgrupp i en hierarki namnet av gruppen med förfäder åtskilda av en punkt, t ex 'Main.West.F1'.

I exemplet nedan är Eric ansvarig för alla system i anläggningen, och definierad på den översta nivån i hierakin. Nils arbetar med den västra delen av anläggningen definieras i systemgruppen 'West'. Carl, slutligen, arbetar med system i F1 delen av anläggningen. Alla systemgrupperna har attributet UserInherit, vilket gör att de ärver användare av sin förälder.



Användare och systemgrupper skapas i administratören.

- Starta administratören med kommandot 'pwra'.
- Gå in i Användardatabasen från menyn, 'File/Open/UserDatabase'.
- Logga in med login kommandot. Öppna kommando-prompten från menyn, 'Functions/Command' och skriv 'login /adm' på kommandoraden. Om systemgruppen 'administrator' finns definierad måste man dessutom addera användarnamn och passerord för en användare definierad i administrator-systemgruppen.
- Gå in i edit mod från menyn, 'Edit/Edit mode'.
  Systemgrupper och användare representeras av objekt av klasserna SystemGroupReg resp UserReg som visas i paletten till vänster. Objekt skapas genom att man väljer en klass i paletten, och därefter klickar med MB2 (mittenknappen) i det högra fönstret. Objekten läggs i en trädstruktur och klickar man på mappen/lövet på ett existerande objekt läggs det nya objeket som första barn, om kan klicka till höger om mappen/lövet läggs det som syskon.
- Skapa en systemgrupp genom att markera 'SystemGroupReg' i paletten till vänster, och klicka med MB2 (mittenknappen) i det högra fönstret. Öppna SystemGroupReg objektet och ange namn och attribut för systemgruppen. Ange fullständigt hierarkinamn, t ex 'Main.West'.
- Skapa en användare genom att markera 'UserReg' i paletten och klicka med MB2 på mappen/lövet till det SystemGroupReg objekt som UserReg objektet ska ligga under. Öppna UserReg objektet och ange anvädarnamn, passerord och privilegier för användaren.
- Spara.
- Logga ut med kommandot 'logout'.

Användardatabasen ligger på katalogen $pwra_db.

25.2 Registrera volymer

Alla volymer i ett nätvärk måste ha ett unikt volymsnamn och volymsidentitet. För att säkerställa detta registreras alla volymer i en global volymslista.

Registreringen görs från administratören:

- Starta administratören med kommandot 'pwra'.
- Gå in i Globala Volymslistan från menyn, 'File/Open/GlobalVolumeList'.
- Logga in som administratör.
- Gå in i editerings läge, 'Edit/Edit mode' i menyn.
  Volymer registreras mha objekt av klassen VolumeReg som visas i paletten till vänster. I paletten finns även $Hier klassen, med vilken man kan ordna VolumeReg objekten hierarktiskt.
- Skapa ett VolumeReg objekt, öppna objektet och ange volymsnamn (= objektsnamn), volymsidentitet och projekt.
- Spara.
- Logga ut med 'logout'-kommandot.

Volymsnamn

Namn på volymen, ska vara unikt och får ha max 31 tecken.

Volymsidentitet

Volymsidentiteten är ett 32-bitar ord specificerat på formen v1.v2.v3.v4, där v1, v2, v3 och v4 är nummer i intervallet 0-255. Beroende på volymens klass, kan numren väljas i olika intervall.

Rotvolymer	0.1-254.1-254-1.254
Användarklassvolymer	0.0.2-254.1-254

Directory volymen har alltid identiteten 254.254.254.253.

25.3 Skapa projekt

Ett projekt är ett antal noder och volymer som delar samma utvecklingsmiljö. Vanligtvis består den av några process- och operatörsstationer som styr en del av anläggningen, men det finns inga begränsningar för storleken av ett projekt. Man kan välja att ha varje nod i var sitt projekt, eller samtliga noder i ett projekt.

- Alla noder i ett projekt (på samma kommunikationsbuss) har länk öppen mellan varandra genom näthanteraren.
- Alla volymer och noder delar samma filkatalogsträd.
- Alla noder måste uppgraderas till ny ProviewR version samtidigt.

En vanlig storlek är 1-10 noder i ett projekt. För många noder ökar nätverksbelastninge och försvårar uppgradering av projektet.

Skapa ett projekt i administratören:

- Starta administratören med kommandot 'pwra'.
- Projektlistan visas när man startar administratören. Den kan även öppnas från menyn (File/Open/ProjectList).
- Logga in som administratör.
- Gå in i edit mode från meny, 'Edit/Edit mode'.
  Projekt representeras av objekt av klassen ProjectReg, som visas i paletten till vänster.
  ProjectReg objekten kan ordnas i hierakier mha $Hier objekt.
- Skapa ett ProjectReg objekt och ange projektnamn, basversion, filkatalog och beskrivning.
- Projektet skapas när man sparar, efter att en verifiering har gjorts. Spara och logga ut.

Projektnamn

Ett projekt har ett projektnamn som identifierar projektet i utvecklingsmiljön. Jämför systemnamnet som definierar projektet i runtimemiljön. Projektnamn och systemnamn är ofta lika, men behöver inte vara det. Ett system kan faktiska ha många projekt i utvecklingsmiljön. När man uppgraderar, eller gör större förändringar i systemet, tar man lämpligen en kopia av projektet och sparar den gående versionen av systemet för att kunna göra mindra ändringar. Kopian skapas under ett nytt projektnamn men den har samma systemnamn.

Base

ProviewR ett är flerversions system, dvs olika versioner av ProviewR kan installeras i samma utvecklingsmiljö, och projekt med olika version kan existera i samma utvecklingsmiljö.
Ett projekt pekar på en bas, t ex V3.4 eller V4.0, när ett projekt skapas väljer man vilket bas projektet ska peka på.

Path

Projektet består av ett filkatalogsträd där databaser, källkodsfiler, arkiv mm lagras.
Path är rot katalogen till det här trädet.

26 Revisioner

En revision är ett tillstånd i utvecklingsmiljön som lagras i ett versionskontrollsystem. Revisionen ska innehålla alla källkodsfiler så att det är möjligt att återskapa en revision. En revision kan användas för att visa det tillstånd som rådde när revisionen skapades, men också för att bygga körbara system från den återskapade koden för test eller för att köra revisionen in produktion.

Det versionskontrollsystem som supportas för närvarande är Git. Ett nytt repository skapas på katalogen $pwrp_root/src när det första revisionen skapas.

Revisionsfönstret

Revisioner hanteras från revisionsfönstret som öppnas från File menyn i konfiguratorn.

Skapa en revision

En ny revision skapas från Functions/New Revision i menyn. Nya revisioner kan endast skapas om den utcheckade revisionen är en ändpunkt på en gren.

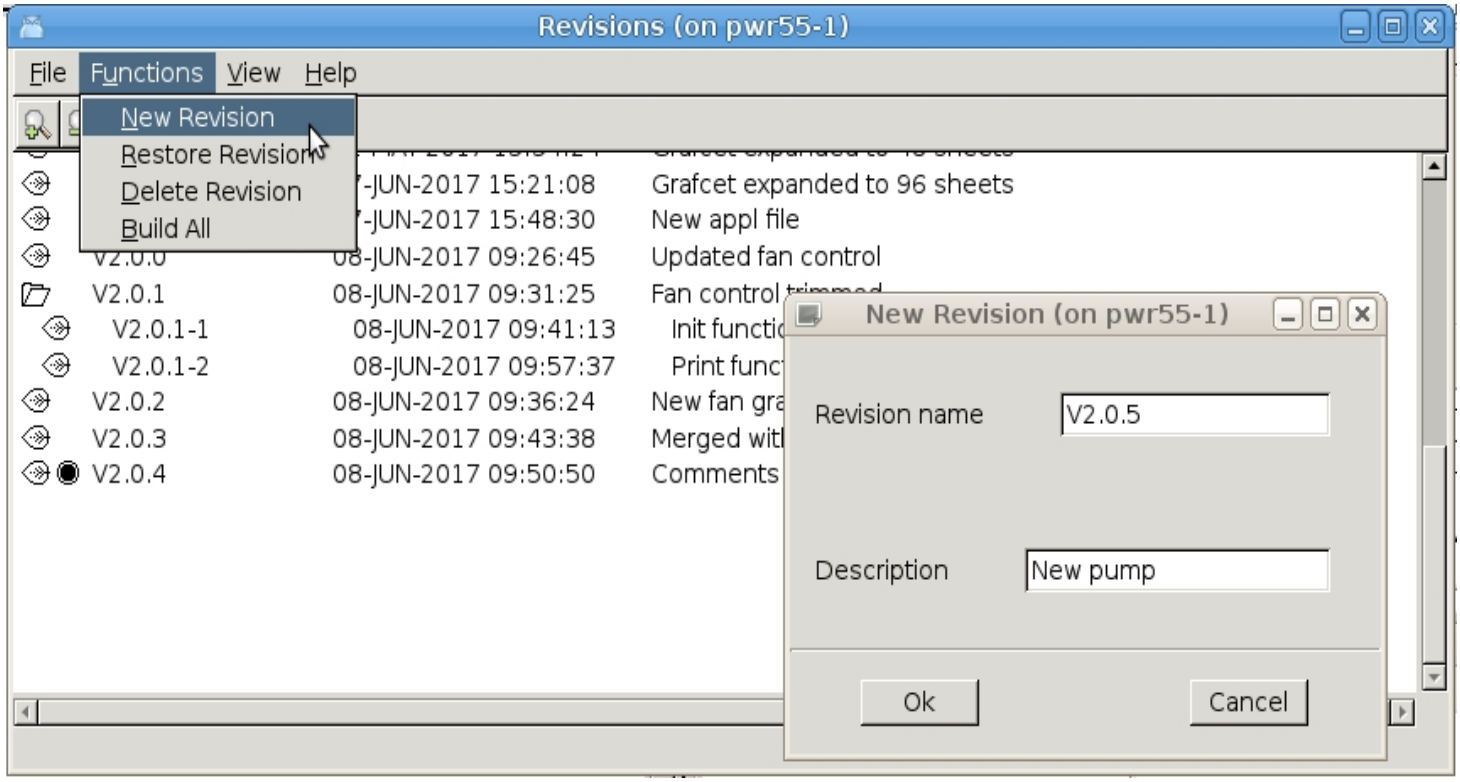


Fig Skapa en ny revision

Återskapa en revision

En revision återskapas från Functions/Restore i menyn. Välj ut den revision som ska återskapas först.

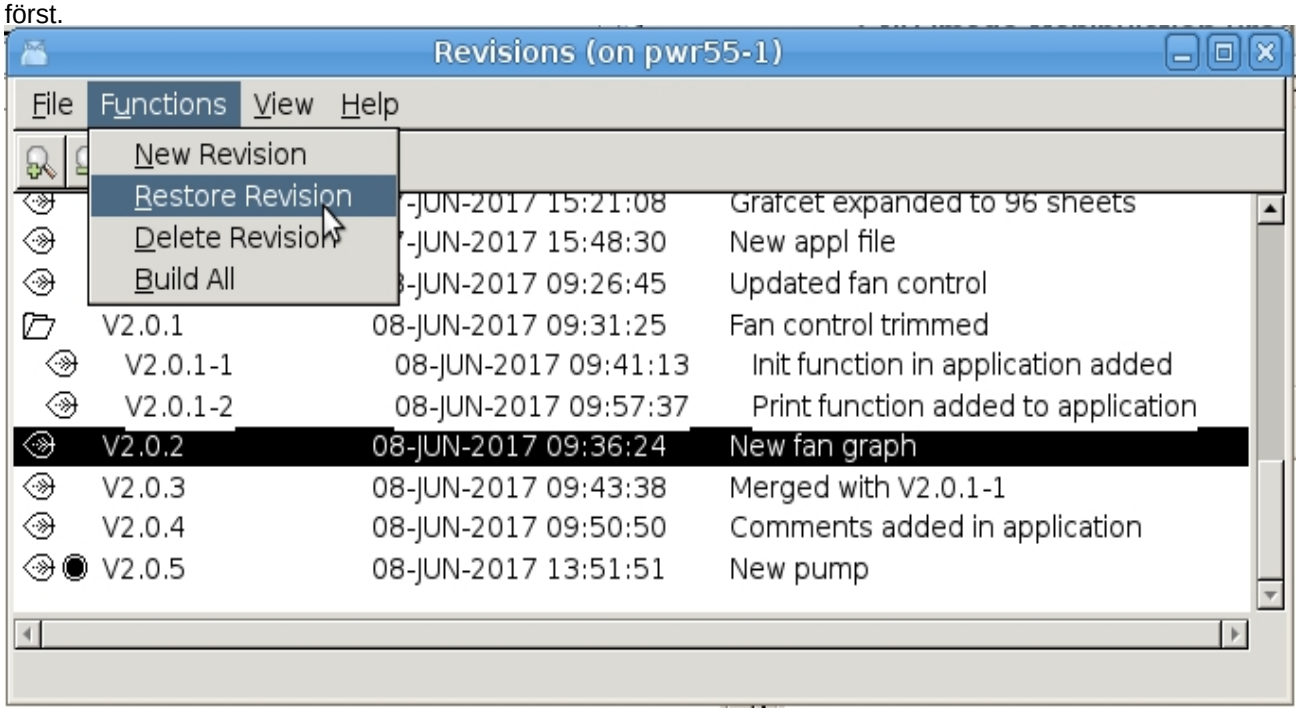


Fig Återskapa en revision

Utcheckad revision markeras med en radioknapp i listan. Även i konfiguratorns titel anges den utcheckade revisionen.

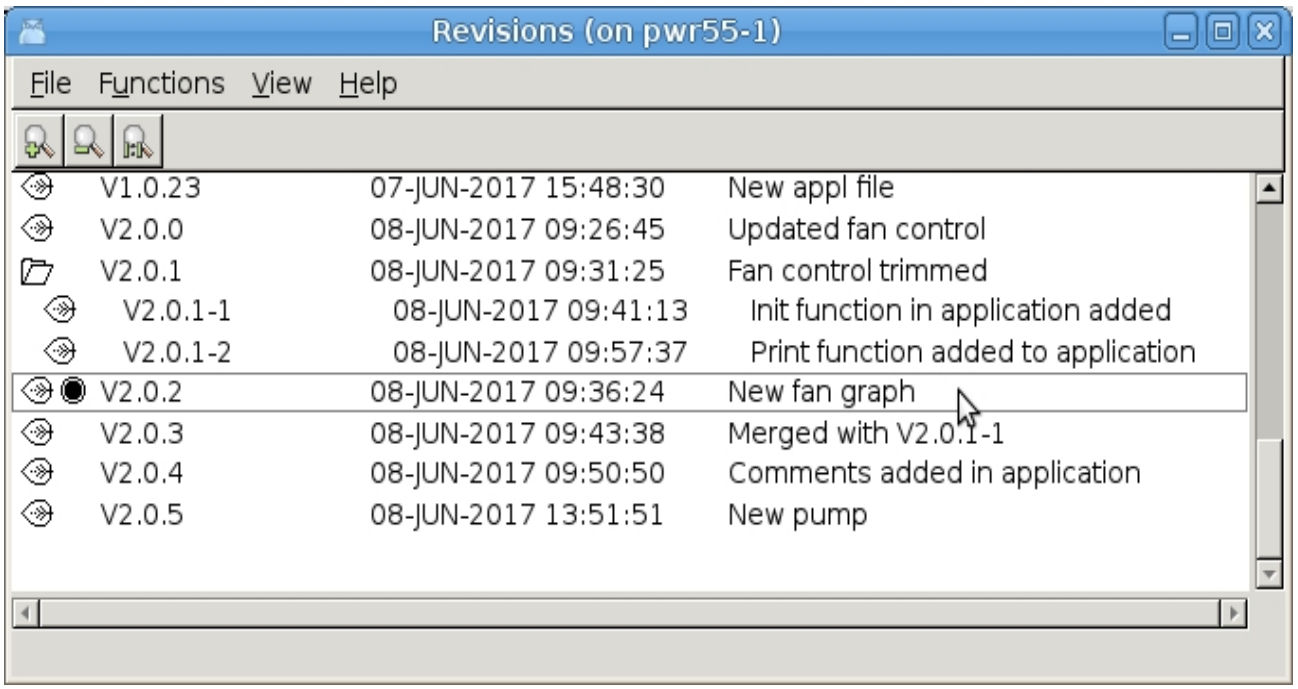


Fig En revision återskapas

Skapa en gren

Om en gammal revision i huvudlinjen återskapas, kan en gren skapas genom att skapa en ny revision från denna punkt. I figuren ovan har en gren skapats från revision V2.0.1. Nya revisioner kan även skapas från ändpunkter av grenar. I figuren ovan, kan en ny revision skapas från V2.0.1-2, som är en ändpunkt, men inte från V2.0.1-1.

Nya filer

Nya filer, t ex c-applikationer, adderas inte automatiskt till git-repositoryt. Detta måste göras manuellt. Nya filer visas av Git som 'untracked files' med 'git status' kommandot. Untracked files kan adderas med 'git add' kommandot och kommer sedan att inkluderas i nästa revision.

OBS! Det är viktigt att addera nya filer för att kunna återskapa en komplett revision.

Exampel

```
> git status
On branch B_V2.0.5
Untracked files:
  (use "git add <file>..." to include in what will be committed)

      appl/ra_appl.c

> git add appl/ra_appl.c
```

27 Verktyg

pwrc	Kommando gränssnitt till utvecklingsdatabasen.
wb_ge	Ge editor
co_help	Hjälp fönster.
pwr_user	Command line interface to user database.
wb_ldlist	Check loadfile versions.

27.1 pwrc

Kommando gränssnitt till utvecklingsdatabasen.

```
pwrc        ProviewR workbench commands

Arguments:
  -v 'volume'    Load volume 'volume'.
  -h             Print usage.

  Other arguments are treated as a command and passed to the command parser
  If a command is given as an argument, the command will be executed and the
  program is then terminated.
  If no command is given, pwrc will prompt for a command.

Examples:
  $ pwrc -v MyVolume
  pwrc>

  $ pwrc -a show volume
  directory     Attached Db  $DirectoryVolume 254.254.254.253
  MyVolume               Db  $RootVolume        0.1.99.20
```

27.2 co_help hjälp fönster

Open a ProviewR help file. By default the project help file is opened.

```
>co_help

Usage:

  co_help [-t 'topic'] [-s 'sourcefile'] [-b 'bookmark']

Arguments:

  -t    Help topic, default 'index'
  -s    Source helpfile
```

```
-b    Bookmark
-l    Language, e.g sv_se
-c    Open Configuration help
-d    Open Designer's Guide
-g    Open Ge Reference Manual
-o    Open Operator Help
```

27.3 wb_ge Ge editor

Startar Ge editorn utan att öppna någon databas.
Användbart om man snabbt vill öppna en Ge graph, men inte ska göra några databas knytningar.

```
> wb_ge ['pwg-file']
```

27.4 pwr_user

pwr_user används för att skapa systemgrupper och användare i användar-databasen. Konfigureringen sker mha kommandon.

pwr_user startas från kommando-prompten.

```
> pwr_user
```

Här beskrivs de olika kommandon som finns för att skapa, modifiera och lista systemgrupper och användare.

add group	Addera en systemgrupp
add user	Addera en användare
get	Hämta en användare
list	Lista systemgrupper och användare
load	Ladda in senaste sparade databas
modify group	Ändra en systemgrupp
modify user	Ändra en användare
remove group	Ta bort en systemgrupp
remove user	Ta bort en användare
save	Spara
su	Logga in som super user

27.4.1 add

add group
add user

27.4.1.1 add group

Skapa en systemgrupp

```
pwr_user> add group 'name' [/nouserinherit]
```

/nouserinherit	Attributet UserInherit ska inte vara satt för systemgruppen. UserInherit är default.

27.4.1.2 add user

Skapa en användare.

```
pwr_user> add user 'name' /group= /password= [/privilege=]
 [/rtread][/rtwrite][/system][/maintenance][/process]
[/instrument][/operator1][/operator2]...[oper10][/devread]
[/devplc][/devconfig][/devclass]
```

/group	Systemgrupp som användaren ska tillhöra.
/password	Användarens passerord.
/privilege	Privilegier om denna anges anges som en mask, dvs ett heltalsvärde.
/rtread	Användaren tilldelas RtRead.
/rtwrite	Användaren tilldelas RtWrite.
/system	Användaren tilldelas System.
/maintenance	Användaren tilldelas Maintenance.
/process	Användaren tilldelas Process.
/operator1	Användaren tilldelas Operator1.
...	
/operator9	Användaren tilldelas Operator9.
/operator10	Användaren tilldelas Operator10.
/devread	Användaren tilldelas DevRead.
/devplc	Användaren tilldelas DevPlc.
/devconfig	Användaren tilldelas DevConfig.
/devclass	Användaren tilldelas DevClass.

27.4.2 get

Hämtar en användare med en algorithm som används i runtime.

```
pwr_user> get 'username' /group= /password=
```

27.4.3 list

Lista systemgrupper och användare.

```
pwr_user> list
```

27.4.4 load

Laddar in den senaste sparade databasen och raderar eventuella ändringar sen senaste save.

27.4.5 modify

modify group
modify user

27.4.5.1 modify group

Ändra data för en systemgrupp.

```
pwr_user> modify group 'name' /[no]userinherit
```

/userinherit	Sätter attributet UserInherit som anger att systemgruppen ska ärva användare från från sin förälder i systemgrupps-hierakin. Negeras med /nouserinherit

27.4.5.2 modify user

Ändra data för en användare.

```
pwr_user> modify user 'name' /group= [/password=][/privilege=]
 [/rtread][/rtwrite][/system][/maintenance][/process]
     [/instrument][/operator1][/operator2]...[oper10][/devread]
     [/devplc][/devconfig][/devclass]
```

/group	Systemgrupp som användaren tillhör.
/password	Användarens passerord.
/privilege	Privilegie om denna anges anges som en mask, dvs ett heltalsvärde.
/rtread	Användaren tilldelas RtRead.
/rtwrite	Användaren tilldelas RtWrite.
/system	Användaren tilldelas System.
/maintenance	Användaren tilldelas Maintenance.
/process	Användaren tilldelas Process.
/operator1	Användaren tilldelas Operator1.
...	
/operator9	Användaren tilldelas Operator9.
/oper10	Användaren tilldelas Operator10.
/devread	Användaren tilldelas DevRead.
/devplc	Användaren tilldelas DevPlc.
/devconfig	Användaren tilldelas DevConfig.
/devclass	Användaren tilldelas DevClass.

27.4.6 remove

remove group
remove user

27.4.6.1 remove group

Ta bort en systemgrupp.

```
pwr_user> remove group 'name'
```

27.4.6.2 remove user

Ta bort en användare.

```
pwr_user> remove user 'name' /group=
```

27.4.7 save

Spara.

```
pwr_user> save
```

27.4.8 su

Logga in som super-user. Som super-user kan man se se password för användare vid listning. su kräver password.

```
pwr_user> su 'password'
```

27.5 wb_ldlist

Visar version för volymer i dbs-filer.
Används för att reda ut version mismatch.

> wb_ldlist <dbs-file>

Exempel

```
wb_ldlist $pwrp_load/volpwrdemo.dbs
Volume     VolPwrDemo      27-MAR-2014 11:06:48.67  0.254.254.200    RootVolume
VolRef     VolPwrDemo      27-MAR-2014 11:06:48.67  0.254.254.200    RootVolume
VolRef     pwrs            14-FEB-2014 16:57:21.19  0.0.0.1          ClassVolume
VolRef     pwrb            14-FEB-2014 16:57:24.82  0.0.0.2          ClassVolume
VolRef     BaseComponent   14-FEB-2014 16:57:52.68  0.0.0.10         ClassVolume
VolRef     OtherManufacturer 14-FEB-2014 16:58:22.73  0.0.250.1      ClassVolume
VolRef     ABB             14-FEB-2014 16:58:10.44  0.0.250.2        ClassVolume
VolRef     Profibus        14-FEB-2014 16:57:34.40  0.0.250.7        ClassVolume
VolRef     Inor            14-FEB-2014 16:58:16.18  0.0.250.8        ClassVolume
VolRef     OtherIO         14-FEB-2014 16:57:55.31  0.0.250.10       ClassVolume
```

28 OPC

ProviewR har implementerat OPC XML/DA protokollet för datautbyte med andra typer av automationenheter. För mer information om OPC, se www.opcfoundation.org.

28.1 OPC XML/DA Server

En OPC XML/DA Server är en web service från vilken an OPC XML/DA Client kan hämta information om ett ProviewR-system. En opc klient kan t ex visa objektshierarkin, läsa och skriva attributvärden, och lägga upp prenumerationer på attribut.

Opc servern implementerar även http protokollet och är inte kopplad till en webserver. Portnummret till opc_server är satt till 80, och URI'n för webservicen på noden 'mynode' blir

```
http://mynode
```

Om en webserver är installerad, använder den vanligtvis port 80, och en annan port måste väljas för opc_server. Om man stället väljer 8080, blir URI'n

```
http://mynode:8080
```

Browsing

Browsing funktionen i OPC XML/DA stödjer grenar (branches) och punkter (item). En punkt innehåller ett värde, medan en gren är en hierarkikomponent utan värde. Det finns inte något stöd för objekt, så ett ProviewR objekt är implementerat som en gren, och varje attribut är en punkt under grenen. Även arrayer är implementerade som grenar, med varje element som en punkt. Om ett element är ett attributobjekt är även detta en gren.

Trådar

Om opc klienten använder HoldTime och WaitTime attributen i SubscriptionPollledRefresh förfrågan, måste opcservern vara multitrådad, dvs skapa en ny tråd för varje förfrågan. Om HoldTime och Waittime inte används (som i ProviewR's opc klient), kan alla förfrågningar hanteras i en enda tråd, vilket sparar cpu-tid. Multitrådning eller inte, konfigureras i konfigurationsobjektet för opc servern. Defaultvärdet är 'IfNeeded' som slår på multitrådning för en klient om HoldTime eller WaitTime upptäcks.

Kient åtkomst

För att få åtkomst till en ProviewR opc server, måste ip-adressen för klienten konfigureras i konfigurationsobjektet för opc servern. Här kan man också välja om klienten ska a läs- eller skriv-rättigheter (ReadOnly resp ReadWrite). ReadOnly tillåter klienten att läsa och prenumerera på värden, medan ReadWrite även tillåter skrivning av attributvärden.

Buffring av prenumerationer

Servern stödjer inte buffring av prenumerationer.

Konfigurering

Opc servern konfigureras med ett Opc_ServerConfig objekt, som placeras i nodhierarkin. Konfigurationsobjektet medför att en serverprocess (opc_server) startas vid ProviewR startup.

28.1.1 OPC XML/DA Client

Proveiw's opc klient är implementerad som en extern volym, som monteras i rotvolymens objektsträd. Under monteringsobjektet visas de grenar och punkter som opc servern innehåller med speciella opc objekt. Ett Opc_Hier objekt representerar en gren, ett Opc_Int objekt en punkt med ett heltalsvärde, ett Opc_Boolean objekt en punkt med ett boolskt värde etc.
Om ett punktobjekt öppnas, visas punkvärdet i Value attributet, och en del andra egenskaper som beskrivning, lowEU, highEU, ingenjörsenhet, lowIR och highIR visas också. När objektet öppnas startas en prenumeration, och värdet uppdateras kontinuerligt. För heltal- och flyttalsvärden finns även en objektsbild som visar en kurva på värdet.

Med en opc klient kan man

- visa grenar och punkter i Xtt, och även visa punkvärden och sätta punktvärden.
- prenumerera på värden och visa dem i en Ge bild.
- hämta upp punktvärden i ett plcprogram, och även skriva punktvärden.

Opc klienten kräver att namn browsing är implementerat i opc servern.

Ge

Ett punkvärde kan visas i en Ge bild genom att använda namnet på punkten i den externa volymen. Till exempel, om monteringsobjektet för den externa volyment är 'Ext-P1', och det lokala namnet för punkten är

```
/P1/Signals/Ai22
```

kommer sinalnamnet i Ge antagligen att bli (detta är beroende av browsing funktionen i servern)

```
Ext-P1-P1-Signals-Ai22.Value##Float32
```

om det är en flyttalspunkt.

Plc

Punktvärden kan även hanteras i plc programm, genom att använda GeExt... och CStoExt... objekt. De objekt som normalt används för att hämta och lagra attributvärden, GetDp, GetAp, StoDp, StoAp etc, kan inte användas efter som det refererade objektets identitet måste vara känt i utvecklingsmiljön, vilket inte är fallet med externa volymer. I Ext objekten, utgörs referenserna av strängar, vilket gör det möjligt att lägga in punktnamnet. För att hämta värdet av punken i föregående exempel, ska man använda ett GetExtFloat32 objekt, och punktnamnet ska vara

```
Ext-P1-P1-Signals-Ai22.Value
```

För att lagra ett värde i en punkt, t ex /P1/Signals/Ao5, använder man en CStoExtFloat32. Det här objektet gör en villkorlig lagring, och enbart på positiv flank på villkoret. Jämför med CStoAp där värdet är lagrat så länge villkoret är sant. Referensnamnet i CStoExtFloat32 objetet blir i det här fallet

```
Ext-P1-P1-Signals-Ao5.Value
```

Klient process

För varje Opc klient/server förbindelse måste en klientprocess startas. Programmet för denna process är opc_provider, som tar argumenten

1. Opc server URL.
2. Extern volume id.
3. Extern volume name.
4. Server identity (optional, default 200).

Konfigurering

Registrering av ExternVolym

Registrera externvolymen i den globala volymslistan (GlobalVolumeList) med volymsnamn och identitet.

Applikationsfil

Addera en rad i applikationsfilen för att starta opc_provider. Här är ett exempel för en opc klient som kopplas upp sig mot opc servern 'http://servernode::8080. Den registrerade externvolymen har namnet MyOpcVolume med volymsidentiteten 0.1.99.55.

```
opc_provider, opc_provider, noload, run, opc_provider, 9, nodebug,
   http://servernode:8080 0.1.99.55 MyOpcVolume
```

Om punkvärden hämtas i plc programmet, ska prioriteten sättas till 4 (sjätte argumentet).

Monteringsobjekt

Skapa ett monteringsobjekt i anläggningshierarkin i rotvolymen, och lägg in objektsidentiten för volymsobjektet i externvolymen i Objekt attributet. I exemplet ovan är denna objid _O0.1.99.55:0.

Tips

Applikationsfilen ligger på $pwrp_load och har namnet

```
$pwrp_load/ld_appl_'nodename'_'busnumber'.txt
```

där nodename är nodens namn och busnumber Qcombusnumret. Om noden är 'mynode' och busnumret är 507, blir filnamnet

```
 $pwrp_load/ld_appl_mynode_507.txt
```

29 Kommandon

build	**Bygg nod, volym eller objekt**
check classes	**Kontrollera om några klasser behöver updateras**
close graph	**Stäng en Ge graf**
compile	**Kompilera plcpgm**
configure card	**Konfigurera ett kort objekt**
connect	**Koppla ihop signal med kanal**
copy	**Kopiera utvalt objektsträd**
copy object	**Kopiera ett objekt**
create bootfiles	**Skapa bootfiler**
create crossreferencefiles	**Skapa korsreferens filer**
create flowfiles	**Skapa flowfiler for plc trace**
create loadfiles	**Skapa laddatafier**
create object	**Skapa ett objekt**
create structfiles	**Skapa structfiler**
create volume	**Skapa en volym**
cut	**Klipp ut objekt**
define	**Definiera en symbol**
delete object	**Ta bort ett objekt**
delete tree	**Ta bort ett objektsträd**
delete volume	**Ta bort en volym**
disconnect	**Koppla ifrån signal och kanal**
display	**Visa ett fönster**
distribute	**Distribuera till operatörs eller processtation**
edit	**Sätt edit mod**
exit	**Stäng wtt**
help	**Visa hjälp**
generate web	**Generera websidor**
list channels	**Lista kanaler**
list descriptor	**Lista från listdescriptor**
list hierarchy	**Lista hieraki**
list plcpgm	**Lista plcpgm**
list signals	**Lista signaler**
login	**Logga in användare**
logout	**Logga ut användare**
move object	**Flytta ett objekt**
new buffer	**Skapa en ny buffer**
one	**Ett fönster**
open buffer	**Öppna buffer väljar fönstret**
open graph	**Öppna en Ge graf**
paste	**Klistra buffer**
print	**Skriv ut plcpgm**
redraw	**Rita om plcpgm**
release subwindow	**Forsätt exekveringen med graf i window objekt**
revert	**Backa sessionen**
save	**Spara sessionen**
search	**Sök**
set advanceduser	**Sätt avancerad användare**

set alltoplevel	**Visa alla topnivå objekt**
set attribute	**Sätt objekt attribut**
set db	**Sätt databas**
set inputfocus	**Sätt inmatings fokus till fönster**
set showalias	**Visa alias namn**
set showattrref	**Visa attribut refrenser**
set showattrxref	**Visa attribut x-referenser**
set showclass	**Visa objektsklassen**
set showdescription	**Visa beskrivning**
set showobjref	**Visa objekt referenser**
set showobjxref	**Visa objekt x-referenser**
set subwindow	**Öppna en graph i ett window objekt**
set subevents	**Inaktivera händelser for window och tabellobjekt**
set template	**Sätt templatevärden på objekt**
set verify	**Verifiera script**
set window	**Sätt fönsterstorlek**
setup	**Wtt setup**
show children	**Visa ett objekts barn**
show license	**Visa licensvillkor**
show object	**Visa ett objekt**
show objid	**Visa objektidentiteten**
show script	**Visa script filer**
show symbol	**Visa en symbol**
show user	**Visa nuvarande användare**
show version	**Visa wtt version**
show volumes	**Visa alla volymer i arbetsbänken**
sort	**Sortera barnen till ett objekt**
two	**Två fönster**
update classes	**Uppdatera klasser**
wb dump	**Dumpa objekt i en textfil**
wb load	**Ladda objekt från textfil**

Symboler

Närliggande ämnen

script

29.1 Kommando build

Anropa byggmetoden för en node, en volym eller ett objekt.

wtt> build node /name= [/force][/debug][/manual][/crossreference]
wtt> build volume /name= [/force][/debug][/manual][/crossreference]
wtt> build classvolume /name= [/force][/debug][/manual][/crossreference]
wtt> build object /name= [/force][/debug][/manual][/crossreference]

/name	Nodnamn, volymsnamn eller objektsnamn.
/force	Kontrollera inga beroenden, bygg allt.
/debug	Bygg med debug, dvs kompilera och länka med debug.
/manual	Bygg endast specificerad enhet.
/crossreferences	Skapa korsreferensfiler. Giltig för byggande av volymer.

29.2 Kommando check classes

Kontrollera om några klasser behöver updateras.

wtt> check classes

29.3 Kommando close graph

Stäng en Ge graf.

wtt> close graph /file=

/file Namn på Ge grafen.

29.4 Kommando compile

Kompilera plcprogram.

Om ingen hierarki, plcpgm eller window anges, kompileras det utvalda plcpgm'et.

wtt> compile [/debug]
wtt> compile /plcpgm= [/debug]
wtt> compile /window= [/debug]
wtt> compile /hierarchy= [/debug][/modified][/from_plcpgm=]
wtt> compile /volume= [/debug][/modified][/from_plcpgm=]
wtt> compile /allvolumes [/debug][/modified][/from_plcpgm=]

/plcpgm	Namn på ett plcpgm objekt som ska kompileras.
/window	Namn på ett plcfönster som ska kompileras.
/hierarchy	Alla plcpgm in den här hierakin kommer att kompileras.
/volume	Volymsnamn. Alla plcpgm i volymen kommer att kompileras.
/allvolumes	Alla plcpgm i alla volymer i arbetsbänken kommer att kompileras.
/debug	Kompilera med debug.
/modified	Kompilera enbart modifierade plcfönster.

29.5 Kommando configure card

Skapa ett kort med kanaler.

wtt> configure card /rack= /cardname= /channelname= /chanidentity=
/chandescription= /table=

/rack	Namn på det rackobjekt som kortet ska tillhöra.
/cardname	Namn på kortet. Sista namnledet.
/channelname	Namn på en kanal. Sista namnledet. Ett '#' kommer att bytas ut mot kanalnumret. T ex /chan=di33## kommer att ge kanalnamnen di3301, di3302... Om det är fler än en kanal. channelname måste innehålla ett '#' tecken.
/chanidentity	Kanalens identitet. Läggs in i Identity attributet för kanalen.
/chandescripton	Kanal beskrivning. Läggs in i Descripton attributet för kanalen.

29.6 Kommando connect

Kopplar ihop en signal med en kanal.

wtt> connect /source= /destination= [/reconnect]

/source	Ett signal eller kanalobjekt.
/destination	Ett signal eller kanalobjekt.
/reconnect	Om source eller destination redan är kopplade kommer de först att ta ner kopplingen.

29.7 Kommando copy

Kopiera utvalda objektträd till en paste buffer.

wtt> copy [/keepreferences]

/keepreferences	Behåll objektsreferenser till objekt utanför det kopierade trädet. Som default kommer dessa att nollställas.
/ignore_errors	Slutför kopieringen trots upptäckta fel.

29.8 Kommando copy object

Kopiera ett objekt eller ett objekträd.

wtt> copy object /source= /destination= /name= [/hierarchy]
[/first] [/last] [/after] [/before]

/source	Objektet som ska kopieras.
/destination	Förälder eller syskon till det skapade objektet.
/name	Namn på det skapade objektet. Sista namnledet.
/hierarchy	Om käll-objektet har barn, kommer även barnträdet att kopieras.
/first	Objektet kommer att läggas in som första barn till destinations objektet.
/last	Objektet kommer att läggas in som sista barn till destinations objektet.
/after	Objektet kommer att läggas in som syskon efter destinations objektet.
/before	Objektet kommer att läggas in som syskon före destinations objektet.

29.9 Kommando create bootfiles

Skapa en ny bootfil.

wtt> create bootfiles /nodeconfig= [/debug]
wtt> create bootfiles /allnodes [/debug]

/nodeconfig	Namn på NodeConfig-objektet för de nod som en bootfil ska skapas för.
/all	Skapa bootfiler för alla noder i projektet.
/debug	Länka plcprogram med debug.

29.10 Kommando create crossreferencefiles

Skapa filer för att visa korsreferenser i xtt och rtt för aktuell volym.

wtt> create crossreferencefiles [/graph] [/simulation]

/graph	Leta efter korsreferenser i Ge grafer.
/simulation	Addera referenser i simuleringsobjekt.

29.11 Kommando create flowfiles

Skapa flowfiler för plc trace.
Plc programmens layout lagras i flowfiler och används i plc trace.

wtt> create flowfiles /plcpgm=
wtt> create flowfiles /hier=
wtt> create flowfiles /all

Kommando för att skapa flowfiler från template plcpgm i en klassvolym
wtt> create flowfiles /template/plcpgm=
wtt> create flowfiles /template/hier=Class

/all	Skapa flowfiler för all plcprogram i volymen. (kan inte användas i en klassvolym, använd /hier=Class istället).
/plcpgm	Skapa flowfiler för specificerat PlcPgm objekt.
/hierarchy	Skapa flowfiler för alla PlcPgm under specificerad hieraki.
/template	Skapa flowfiler för PlcTemplate program i en klassvolym.

29.12 Kommando create loadfiles

Skapa laddatafiler för en volym.

wtt> create loadfile /volume=
wtt> create loadfile [/class] [/all]

/volume	Skapa laddatafil för angiven volym.
/all	Skapa laddatafiler för alla rot- och sub-volymer i arbetsbänken.
/class	Skapa laddatafiler för alla classvolymer i arbetsbänken.

29.13 Kommando create object

Skapa ett objekt.

wtt> create object /destination= /name= /class=
[/first] [/last] [/after] [/before]

/destination	Destinationen för det nya objektet. Det nya objektet kommer att bli barn eller syskon till destinations objektet.
/name	Namn på det nya objektet. Sista namnledet.
/class	Klass på det nya objektet.
/first	Objektet kommer att läggas in som första barn till destinations objektet.
/last	Objektet kommer att läggas in som sista barn till destinations objektet.
/after	Objektet kommer att läggas in som syskon efter destinations objektet.
/before	Objektet kommer att läggas in som syskon före destinations objektet.

29.14 Kommando create structfiles

Skapa c include filer för klasser i en klassvolym.

wtt> create structfiles [/files=]

/files	Namn på wb_load filen.

29.15 Kommando cut

Kopiera utvalda objektträd till en paste buffer, och ta bort objekten
ur aktuell volym.

wtt> cut [/keepreferences]

/keepreferences	Behåll objektsreferenser till objekt utanför det kopierade trädet. Som default kommer dessa att nollställas.

29.16 Kommando define

Definiera en symbol.

wtt> define 'symbolname' 'text'

närliggand ämnen

symbol
show symbol
symbolfile

29.17 Kommando delete object

Ta bort ett objekt.

wtt> delete object /name= [/noconfirm] [/nolog]

/name	Namn på objektet.
/noconfirm	Ta bort utan verifikation.
/nolog	Operationen kommer inte att loggas.

29.18 Kommando delete tree

Ta bort ett objektsträd.

wtt> delete tree /name= [/noconfirm] [/nolog]

/name	Rot objektet för trädet.
/noconfirm	Ta bort utan verifikation.
/nolog	Operationen kommer inte att loggas.

29.19 Kommando disconnect

Ta ner kopplingen för en signal eller kanal.

wtt> disconnect /source=

/source	Ett singal eller kanal objekt.

29.20 Kommando display

Visa anläggnings eller nod hierarkin i fönster (w1 eller w2).

wtt> display w1
wtt> display w2

29.21 Command distribute

Distribuerar filer till operatörs och processnoder.
Skapar ett distributions paket, kopierar paketet till
stationen och packar upp det.

wtt> distribute /node= [/package] [/file=]

/node	Nod att distribuera till.
/package	Skapar enbar ett paket. Paketet skapas men kopieras inte.
/file	Namn på paket. Kopierar angivet paket utan att skapa ett nytt.

29.22 Kommando edit

Starta eller lämna editerings mod.

wtt> edit
wtt> noedit

29.23 Kommando exit

Close wtt.

wtt> exit

29.24 Kommando help

Visa hjälp information för ämnet.
Hjälp informationen söks efter i en hjälpfil. Filen kan vara
bas-hjälpfilen, projekt-hjälpfilen eller en annan hjälpfil.

Om ingen hjälpfil anges kommer ämnet att eftersökas i
bas och projekt hjälpfilerna.

wtt> help 'subject'
wtt> help 'subject' /helpfile=

/helpfile	En hjälpfil som innehåller information om ämnet.

närliggande ämnen

helpfile

29.25 Kommando generate web

Generera html-filer för websidor konfiguerarde med Web-objekt i nodhierarkin för aktuell volym.

wtt> generate web

29.26 Kommando list

Skriv ut en lista på objekt och attribut.

Listan kommer att skickas till en skrivarkö specificerad av symbolen PWR_FOE_PRINT.

wtt> list descriptor /descriptor=
wtt> list channels [/node=]
wtt> list signals [/hierarchy=]
wtt> list plcpgm [/plcpgm=] [/hierarchy=]
wtt> list hierarchy [/hierarchy=]

29.27 Kommando list channels

Lista kort och kanaler.

wtt> list channels [/node=] [/volume=] [/allvolumes] [output=]

/node	$Node objekt.
/volume	Lista objekt i angiven volym.
/allvolume	List objekt i alla volymer.
/output	Filnamn för utskrift på fil. Om ett filnamn anges kommer listan inte att skickas till en printer.

29.28 Kommando list descriptor

Skriver ut en lista beskriven av ett ListDescriptor objekt.

wtt> list descriptor /descriptor=

/descriptor ListDescriptor object.

29.29 Kommando list hierarchy

Lista PlantHier och NodeHier objekt.

wtt> list hierarchy [/hierarchy=] [/volume=] [/allvolumes] [output=]

/hierarchy	Hierarki objekt.
/volume	Lista objekt i angiven volym.
/allvolume	List objekt i alla volymer.
/output	Filnamn för utskrift på fil. Om ett filnamn anges kommer listan inte att skickas till en printer.

29.30 Kommando list plcpgm

Lista PlcPgm objekt.

wtt> list plcpgm [/hierarchy=] [plcpgm=] [/volume=] [/allvolumes] [output=]

/plcpgm	Plcpgm objekt.
/hierarchy	Hierarki objekt.
/volume	Lista objekt i angiven volym.
/allvolume	List objekt i alla volymer.
/output	Filnamn för utskrift på fil. Om ett filnamn anges kommer listan inte att skickas till en printer.

29.31 Kommando list signals

Lista signaler och korsreferenser för signalerna.

wtt> list signals [/hierarchy=] [/volume=] [/allvolumes] [output=]

/hierarchy	Hierarki objekt.
/volume	Lista objekt i angiven volym.
/allvolume	List objekt i alla volymer.
/output	Filnamn för utskrift på fil. Om ett filnamn anges kommer listan inte att skickas till en printer.

29.32 Kommando login

Logga in med användarnamn och passerord. Användarens privilegier kommer att hämtas ur användardatabasen, och påverka tillgången till systemet.

wtt> login 'username' 'password'

Om man vill skapa eller modifiera projekt, användare eller registrera volymer loggar man in som administratör med kvalifieraren /administrator. Här måste man ange en användare i systemgruppen 'administrator'. Om denna systemgrupp ej finns, kan användarnamn och passerord utelämnas.

wtt> login /administrator 'username' 'password'

närliggande ämnen

logout
show user

29.33 Kommando logout

Logga ut en användare, och återgå till den ursprungliga användaren.

wtt> logout

närliggande ämnen

login

29.34 Kommando move object

Flytta eller byt namn på ett objekt.

wtt> move object /source= /destination= [/rename=] [/first] [/last] [/after] [/before]
wtt> move object /source= /rename=

/source	Namn på objektet som ska flyttas.
/destination	Förälder eller syskon objekt efter förflyttningen.
/rename	Nytt objektnamn, om namnet ska ändras. Sista namnledet. Om ingen destination anget kommer enbart namnet att ändras, objektet flyttas inte.
/first	Objektet kommer att läggas in som första barn till destinations objektet.
/last	Objektet kommer att läggas in som sista barn till destinations objektet.
/after	Objektet kommer att läggas in som syskon efter destinations objektet.
/before	Objektet kommer att läggas in som syskon före destinations objektet.

29.35 Kommando new buffer

Skapa en ny tom buffer.

wtt> new buffer /name=

/name Namn på bufferten.

29.36 Kommando one

Visa ett fönster. Fönstret som för närvarande har inmatnings fokus behålls.

wtt> one

29.37 Kommando open buffer

Öppna fönstret för att välja buffer.

wtt> open buffer

29.38 Kommando open graph

Öppna en Ge graf.
I modal mod, fortsätter inte exekveringen av skriptet förrän grafen stängs.

wtt> open graph /file= /modal

/file	Namn Ge grafen.
/modal	Modal mod.

29.39 Kommando paste

Klistra in objekt från den senaste copy or cut operationen, i den nuvarande volymen. Med buffer möjligheten kan en äldre buffer klistras in.

wtt> paste [/keepoid] [/buffer=]

/keepoid	Behåll objektsidentiteter om det är möjligt.
/buffer	Namn på den buffer som ska klistras in. Som default används den senaste bufferten.
/into	Lägg rotobjekten i pastebufferten som barn till utvalt objekt.
/toplevel	Lägg rotobjekten i pastebufferten på topnivån. Måste användas för att kopiera till en tom volym.

29.40 Kommando print

Skriv ut plc dokument.

wtt> print /plcpgm= [/nodocument] [/nooverview]
wtt> print /hierarchy= [/nodocument] [/nooverview]

/plcpgm	Skriv ut dokument i ett plcpgm.
/hierarchy	Ett hierarki objekt. Alla plc i hierarkin kommer att skrivas ut.
/nodocument	Plc dokumentet skrivs ej ut.
/nooverview	Översikten över plc-fönstret skrivs ej ut.
/pdf	Skriv till pdf-fil.
/all	Skriv ut alla plcpgm.

29.41 Kommando redraw

Rita om plc-koden.

wtt> redraw /all
wtt> redraw /hierarchy=
wtt> redraw /plcpgm=

/plcpgm	Rita om ett plcpgm.
/hierarchy	Hierarkiobjekt. Alla plcpgm i hierarkin kommer att ritas om.
/all	Rita om alla plcpgm.

29.42 Kommando release subwindow

Släpp loss exekveringen av ett skript som öppnat en graf
i ett window objekt med kommandot 'set subwindow' eller
funktionen 'SetSubwindow' i modal mod.
Release kommandot exekveras från en trycknapp i bilden med
actiontyp command.

wtt> release subwindow 'graph'

graph	Namnet på huvud grafen.

29.43 Kommando revert

Revertera sessionen.

wtt> revert

29.44 Kommando save

Spara sessionen.

wtt> save

29.45 Kommando search

Leta efter ett objektsnamn eller en sträng.

wtt> search 'object'
wtt> search /regularexpression 'expression'
wtt> search /next

29.46 Kommando set advanceduser

Sätta eller återställa 'advanced user'.

wtt> set advanceduser
wtt> set noadvanceduser

närliggande ämnen

advanced user

29.47 Kommando set alltoplevel

Visa alla rotobjekt i databasen, inte enbart de som är definierade för anläggnings resp node hierarkin.

wtt> set alltoplevel
wtt> set noalltoplevel

29.48 Kommando set attribute

Sätt ett värde på ett attribut.
Objekt väljs med name, class och hierarchy kvalifierarna.

wtt> set attribute /attribute= [/value=] [/name=] [/class=] [/hierarchy=]
[/noconfirm] [/nolog] [/output] [/noterminal]

/attribute	Namn på attributet.
/value	Värde som ska läggas in i attributet. Om inget värde är angivet kommer en fråga för varje attribut.
/class	Välj objekt av den här klassen.
/hierarchy	Endast avkomlingar till det här objektet kommer att väljas.
/noconfirm	Ställ ingen verifikations fråga.
/nolog	Operationen loggas inte.
/output	Ut fil.
/noterminal	Operationen loggas inte i terminal fönster.

Exempel

```
wtt> set attribute /name=H1-Pump /attr=Description /value="Water pump" /noconf
```

29.49 Kommando set db

Koppla upp till databasen med angivet id.
Detta har ingen effekt om man redan har öppnat en databas.

wtt> set db /dbid=

/dbid	Databas identitet.

29.50 Kommando set inputfocus

Sätt inmatnings fokus till anläggnings eller nod hierarki fönstret
(w1 eller w2).

wtt> set inputfocus w1
wtt> set inputfocus w2

29.51 Kommando set showalias

Visa aliasnamnet för objekt i anläggnings och nod hierakin.

wtt> set showalias
wtt> set noshowalias

29.52 Kommando set showattrref

Visa antal kopplade attribut referenser för objekt
i anläggnings och nod hierarkin.

wtt> set showattrref
wtt> set noshowattrref

29.53 Kommando set showattrxref

Vissa antalet kopplade attribut x-referenser för objekt
in anläggnings och nod hierarkin.

wtt> set showattrxref
wtt> set noshowattrxref

29.54 Kommando set showclass

Visa klasstillhörighet för objekt i anläggnings och nod hierarkin.

wtt> set showclass
wtt> set noshowclass

29.55 Kommando set showdescription

Visa beskrivning för objekt i anläggnings och nod hierarkin.

wtt> set showdescription
wtt> set noshowdescription

29.56 Kommando set showobjref

Visa antalet kopplade objektreferenser för objekt i anläggnings och nod hierarkin.

wtt> set showobjref
wtt> set noshowobjref

29.57 Kommando set showobjxref

Visa antalet kopplade objekt x-referenser för objekt i anläggnings och nod hierarkin.

wtt> set showobjxref
wtt> set noshowobjxref

29.58 Kommando set subwindow

Öppna en graf i ett window objekt i en tidigare öppnad graf, eller
skifta graf i en multiwindow cell.

wtt> set subwindow 'graph' /name= /source=
wtt> set subwindow 'multiview-object' /name= /source= [/continue]

'graph'	Graf där windowobjektet finns. '$current' anger nuvarande graf.
/name	Namn på window objektet.
/source	Namn på grafen som ska öppnas i window objektet.
/object	Anger objekt om grafen är en objektsgraf.
/x0	Övre vänstra x-koordinat för grafens gräns. Om x0, y0, x1, y1 anges, kommer en annan del än den som är angiven i Graph attributes x0, y0, x1, y1 att visas. Implementerad för grafer i multiview celler, inte för window objekt.
/y0	Övre vänstra y-koordinat för grafen gräns.
/x1	Undre högra x-koordinat för grafen gräns.
/y1	Undre högra y-koordinat för grafen gräns.
/continue	Kan användas om kommandot exekveras från en Ge knapp med annan dynamik som ska exekveras. Ska inte användas om grafen med knappen själv skiftas.

Om 'set subwindow' kommandot används i ett script som ersätter nuvarande graf, måste scriptet avslutas med ett anrop till exit(GLOW__SUBTERMINATED).

```
main()
  set subwindow motor1 /name=subw1 /source=motor1.pwg
  ...
  exit( GLOW__SUBTERMINATED);
endmain
```

29.59 Kommando set template

Föråldrad sen V4.0.

29.60 Kommando set verify

Visa alla exekverade rader när ett script körs.

wtt> set verify
wtt> set noverify

29.61 Kommando set window

Set fönsterbredd och höjd.

wtt> set window /width= /height=

/width	bredd i pixel.
/height	höjd i pixel.

29.62 Kommando set volume

set volume är obsolet.

29.63 Wtt setup

Setup av wtt egenskaper

DefaultDirectory	Default filkatalog för script filer.
SymbolFilename	Symbolfil.
Verify	Verifiera exekvering av script.
AdvancedUser	Användaren är avancerad.
AllToplevel	Visa samtliga topnivå objekt.
Bypass	Förbigå vissa editerings restriktioner.

29.64 Kommando show children

Visa ett objekt och dess barn.

wtt> show children /name=

/name	Förälder objektets namn.

29.65 Kommando show license

Visa licensvillkor.

wtt> show license

29.66 Kommando show object

Lista objekt.

wt> show object [/name=] [/hierarchy=] [/class=] [/volume=] [/allvolumes]
[/parameter=] [/full] [/output=] [/noterminal]
wt> show object /objid=

/name	Objektsnamn. Wildcard kan användas.
/hierarchy	Hierarki objekt. Endast objekt i hierarkin kommer att väljas.
/class	Endast objekt av angiven klass kommer att väljas.
/volume	Namn på volymen.
/allvolumes	Sökning av objekt kommer att ske i alla volymer.
/parameter	Lista värdet på ett attribut för valda objekt.
/full	Visa innehållet för objekten. Attribut som skiljer sig från template värdet kommer att listas.
/output	Ut fil.
/noterminal	Resultatet kommer inte att skrivas i terminalfönstret.
/objid	Visa objekt med specificerad objektsidentitet.

29.67 Kommando show objid

Visa objektsidentitet för ett objekt.
Om /name utelämnas, visas identiteten för det utvalda objektet.

wtt> show objid [/name=]

/name	Objektsnamn.

29.68 Kommando show script

Visar en lista på script-filer.
Wildcard med asterisk (*) kan användas för att ange filer.

wtt> show script ['scriptspec']

29.69 Kommando show symbol

Visa en symbol, eller alla symboler.

wtt> show symbol 'symbol' **Visa symbol 'symbol'**
wtt> show symbol **Visa alla symboler**

närliggande ämnen

define
symbol

29.70 Kommando show version

Visa wtt version.

wtt> show version

29.71 Kommando show volumes

Visa alla volymer i arbetsbänken.

wtt> show volumes

29.72 Kommando sort

Sortera barnen til ett objekt i alfabetisk ordning, eller i klassordning.
Om ingen 'parent' anges kommer barnen till utvalt objekt att sorteras.

wtt> sort /parent= [/class] [/signals]

/parent	Förälder till de objekt som ska sorteras.
/class	Sortera i klass ordning.
/signals	Sortera signaler och plcpgm i klassordning, och andra objekt i alfabetisk ordning.

29.73 Kommando two

Visa två fönster. Både anläggnings och nod hierarki fönsterna visas.

wtt> two

29.74 Kommando update classes

Updatera klasserna i aktuell volym.

wtt> update classes

29.75 Kommando wb dump

Dumpa volymen, eller en del av volymen på textfil.

wtt> wb dump /output= [/hierarchy=]

/hierarchy	Hierarki objekt. Objektet och dess barnträd kommer att dumpas på textfil.
/output	Utfil.
/nofocode	Skriv inte plc kod för funktionsobjekt med template kod. Reducerar storleken på dumpfilen. Ny kod kommer att kopieras när plc't kompileras.
/keepname	Skriv externa referenser med namn istället för identitet.
/noindex	Skriv inte objektindex i dumpfilen.

29.76 Kommando wb load

Ladda databasen från wb_load-fil eller dbs-fil.

wtt> wb load /loadfile=

/loadfile	Filnamn. Kan vara av typ .wb_load, .wb_dmp eller .dbs.
/noindex	Ignorera object index i dumfilen och skapa nya objektsidentiteter.

29.77 Symbol

En wtt symbol kan avändas som ett kort kommando eller som strängsubstitution i ett kommando.
Då symbolen används som strängsubstitution ska symbolnamnet omges av enkelfnuttar.

Symboler skapas med 'define' kommandot.
Define kommandon kan exekveras av konfiguratörens startupfil.

Exempel på symbol som används som kort kommando.

```
wtt> define p1 "show child/name=hql-hvk-pumpar-pump1"
wtt> p1
```

Exempel på symbol som används som sträng substitution

```
wtt> define p1 hql-hvk-pumpar-StartPump1
wtt> open trace 'p1'
```

Närliggande ämnen

define
show symbol
symbolfile

30 Wtt script

Exekvera ett script

Datatyper och declarationer

Datatyper
Datatyps konvertering
Variabel deklarationer
Operatorer

Uttryck

main-endmain
function-endfunction
if-else-endif
while-endwhile
for-endfor
break
continue
goto
include

In och utmatnings funktioner

ask()
printf()
say()
scanf()

Filhanterings funktioner

fclose()
felement()
fgets()
file_search()
fopen()
fprintf()
fscanf()
translate_filename()

Sträng funktioner

edit()
element()
extract()
sprintf()
strchr()
strrchr()
strlen()
strstr()
toupper()

tolower()

Databas funktioner

GetAttribute()
GetChild()
GetParent()
GetNextSibling()
GetNextVolume()
GetClassList()
GetNextObject()
GetClassListAttrRef()
GetNextAttrRef()
GetTemplateObject()
GetNextTemplateAttrRef()
GetObjectClass()
GetNodeObject()
GetRootList()
GetVolumeClass()
GetVolumeList()
SetAttribute()
CreateObject()
RenameObject()
MoveObject()
InLib()
OpenPlcPgm()
ClosePlcPgm()
CreatePlcObject()
CreatePlcConnection()
SetPlcObjectAttr()
PlcConnect()

System funktioner

exit()
get_namespace()
set_namespace()
system()
terminate()
time()
tzset()
verify()

Diverse funktioner

GetProjectName()
CheckSystemGroup()
CutObjectName()
MessageError()
MessageInfo()
GetCurrentText()
GetCurrentObject()
GetCurrentVolume()
IsW1()
IsW2()
EditMode()
MessageDialog()
ConfirmDialog()

ContinueDialog()
PromptDialog()
OpenGraph()
CloseGraph()
SetSubwindow()
GetIoDeviceData()
SetIoDeviceData()
GetVersion()
get_pwr_config()
get_node_name()
getmsg()
EVEN()
ODD()

wtt-kommandon

wtt-kommandon

30.1 Exekvera ett script

En script-fil exekveras från kommando-raden med kommandot

wtt> @'filnamn'

30.2 Datatyper

Datatyperna är float, int och string.

int	heltals värde.
float	32-bitars flyttals värde.
string	textsträng, 80 tecken lång (null terminerad).

Det finns tre olika tabeller som en variabel kan deklareras i: local, global och extern. En lokal variabel är känd inom en funktion, en global är känd i alla funktioner inom en fil, en extern är känd i alla filer som exekveras i en session.

30.3 Datatyps konvertering

Om ett uttryck består av variabler och funktioner med olika datatyper kommer variablerna att konverteras med företräde string, float, int. Om två operander har typerna float och string, eller int och string, kommer resultatet att bli string. Vid en tilldelning, kommer värdet av ett uttryck att konverteras till typen för den tilldelade variabeln. Detta gäller även om resultatet är en sträng och variablerna är av type float och int.

Exempel

```
string str;
int    i = 35;
str = "Luthor" + i;
```
The value in str will be "Luthor35".

```
float   f;
string  str = "3.14";
int     i = 159;
f = str + i;
```
The value in f will be 3.14159.

30.4 Variabel deklarationer

En variabel måste deklareras innan den avänds.
En deklaration består av
- tabellen (global eller extern, om local ska inte tabellen anges)
- datatypen (int, float eller string)
- variabelnamn (känsligt för stora och små bokstäver)
- antal element, om variabeln är en vektor
- likameds tecken följt av initierings värde, om det utelämnas är intitialvärdet 0 eller en null-sträng
- semikolon.

En extern variabel bör tas bort (med delete) när den inte längre används.
Även globala variabler kan tas bort med deletegbl.

Exempel

```
int             i;
float           flow = 33.4;
string          str = "Hello";
float           width[5] = (1.20, 2.44, 4.81, 7.77, 9.20);
extern int      jakob[20];
global float    ferdinand = 1234;
...
delete jakob[20];
detetegbl ferdinand;
```

Namnrymd

Externa variabler kan deklareras i olika namnrymd genom att sätta namnrymd med set_namespace(). Det kan användas för att exekvera uppsättningar av script samtidigt i samma process. En uppsättning script kan hantera gemensamma externa variabler, och genom att använda olika namnrymder kan samma uppsättning exekveras i flera versioner utan sammanblandning av de externa variablerna.

30.5 Operatorer

Operatorerna har samma funktion som i c, med vissa begränsningar. Alla operatorer är inte implementerade. Vissa operatorer (+,=,==) kan även

operera på string variabler. Prioriteten för operatorer är densamma som i c.

Operator	Beskrivning	Datatyper
+	plus	int, float, string
-	minus	int, float
*	multiplikation	int, float
/	division	int, float
++	inkrement, postfix only.	int, float
--	dekrement, postfix only	int, float
>>	bitar höger-skift	int
<<	bitar vänster-skift	int
<	mindre än	int, float
>	större än	int, float
<=	mindre eller lika med	int, float
>=	större eller lika med	int, float
==	lika med	int, float, string
!=	ej lika med	int, float, string
&	bitvis och	int
|	bitvis eller	int
&&	logisk och	int
||	logisk eller	int
!	logisk not	int
=	tilldelning	int, float, string
+=	addera och tilldela	int, float
-=	minus och tilldela	int, float
&=	logisk och och tilldelning	int
|=	logisk eller och tilldelning	int

30.6 Script uttryck

main-endmain	Main funktion.
function-endfunction	Funktions deklaration.
if-else-endif	Villkorlig exekvering.
while-endwhile	While loop.
for-endfor	For loop.
break	Avsluta while eller for loop.
continue	Fortsätt while eller for loop.
goto	Hoppa till label.
include	Inkludera script fil.

30.6.1 main-endmain

main och endmain satserna kontrollerar var exekveringen börjar och slutar. Om det inte finns några main och endmain satser, startar exekveringen i början på filen, och slutar i slutet på filen.

Exempel

```
main()
  int a;

  a = p1 + 5;
  printf( "a = %d", a);
endmain
```

30.6.2 function-endfunction

En funktions deklaration består av
- datatypen för funktionens retur värde
- namnet på funktionen
- en argumentlista separerade med kommatecken och omgiven av parenteser.
  Argumentlistan innehåller typdeklaration och namn på varje argument.

Argumenten som skickas med vid anropet kommer att konverteras till den typ som är deklarerad i argumentlistan. Om ett argument ändrar värde inne i funktionen, kommer det nya värdet att överföras till anroparen. På detta sätt blir de möjlig att returnera andra värden än retur värdet för funktionen. En funktion kan innehålla en eller flera return statser. Return satsen kommer att flytta över exekveringen till anroparen och returnera det angivna värdet.

Exempel

```
function float calculate_flow(float a, float b)
  float c;
  c = a + b;
  return c;
endfunction

...
flow = korr * calculate_flow( v, 35.2);
```

30.6.3 if-else-endif

Raderna mellan en if-endif stats exekveras om uttrycket i if-statsen är sant. Uttrycket ska omges av parenteser. Om en else sats hittas mellan if och endif, kommer raderna mellan else och endif att exekveras när uttrycket är falskt.

Exempel

```
if ( i < 10 && i > 5)
  a = b + c;
endif

if ( i < 10)
  a = b + c;
else
  a = b - c;
endif
```

30.6.4 while-endwhile

Raderna mellan en while-endwhile sats exekveras så länge uttrycket i while-satsen är sant. Uttrycket ska omges av parentser.

Exempel

```
while ( i < 10)
  i++;
endwhile
```

30.6.5 for-endfor

Raderna mellan en for-endfor sats exekveras så länge mitten-uttrycket i for-satsen är sant. for-satsen består av tre uttryck, avgränsade med semicolon och omgivna av parenteser. Det första uttrycket exekveras före den första loopen, det tredje exekveras efter varje loop, och det i mittne exekveras före varje loop, och om det är sant, gör ytterligare ett varv, annars lämnas loopen.

Exempel

```
for ( i = 0; i < 10; i++)
  a += b;
endfor
```

30.6.6 break

En break sats kommer att söka efter nästa endwhile eller endfor sats och försätta exekveringen på raden efter denna.

Exempel

```
for ( i = 0; i < 10; i++)
  a += b;
  if ( a > 100)
    break;
endfor
```

30.6.7 continue

En continue sats kommera att söka efter närmast föregående while eller for sats och forsätta att utföra loopen.

Exempel

```
for ( i = 0; i < 10; i++)
  b = my_function(i);
  if ( b > 100)
    continue;
  a += b;
endfor
```

30.6.8 goto

En goto sats orsakar ett hopp i exekveringen till en rad som är definierad med en label. Label raden avslutas med ett kolon.

Exempel

```
  b = attribute("MOTOR-ON.ActualValue", sts);
  if (!sts)
    goto some_error;
  ...
some_error:
  say("Something went wrong!");
```

30.6.9 include

En script include-fil xom innehåller funktioner kan inkluderas med #include satsen. Default filtyp för filen är '.pwr_com'.

Exempel

```
#include <my_functions>
```

30.7 In och utmatnings funktioner

Funktion	Beskrivning
ask	Skriv en fråga och läs ett svar.
printf	Formaterad utskrift.
say	Skriv en text.
scanf	Formaterad läsning.

30.7.1 ask()

int ask(string question, (arbitrary type) reply)

Beskrivning

Promptar för inmatning med den angivna strängen.
Returnerar antal lästa element, 1 or 0.

Arguments

string	question	Prompt.
godtycklig typ	reply	Inmatat svar. Kan vara int, float eller string.

Exempel

```
 string reply;

 ask( "Do you want to continue? [y/n] ", reply);
 if ( reply != "y")
   exit();
 endif
```

30.7.2 printf()

int printf(string format [, (arbitrary type) arg1, (arbitrary type) arg2])

Beskrivning

Formaterad utskrift. C-syntax. Format argument och inget, ett eller två värde argument.
Returnerar antal utskrivna tecken.

Argument

string	format	Format.
godtycklig typ	arg1	Värde argument. Valfritt, Kan vara int, float eller string.
godtycklig typ	arg2	Värde argument. Valfritt. Kan vara int, float eller string.

Exempel

```
printf( "Watch out!");
printf( "a = %d", a);
printf( "a = %d och str = %s", a, str);
```

30.7.3 say()

int say(string text)

Beskrivning

Skriver ut en sträng.

Argument

string	text	Text att skriva ut.

Exempel

```
say( "Three quarks for Muster Mark!");
```

30.7.4 scanf()

int scanf(string format , (godtycklig typ) arg1)

Beskrivning

Formaterad inmatning. C-syntax.
Returnerar antal inlästa tecken.

Argument

string	format	Format.
godtycklig typ	arg1	Värde argument. Returnerat. Kan vara int, float eller string.

Exempel

```
scanf( "%d", i);
```

30.8 Filhanterings funktioner

Funktion	Beskrivning
fclose	Stäng en fil.
felement	Extrahera ett element från senaste lästa rad.
fgets	Läs en rad från en fil.
file_search	Sök efter filer.
fopen	Öppna en fil.
fprintf	Formaterad utskrift till fil.
fscanf	Formaterad läsning från fil.
translate_filename	Översätt omgivnings-variabler i ett filnamn.

30.8.1 fclose()

int fclose(int file)

Beskrivning

Stänger en öppnad fil.

Argument

int	file	fil-id returnerad av fopen.

Exempel

```
int infile;
infile = fopen("some_file.txt","r");
...
fclose( infile);
```

30.8.2 felement()

string felement(int file int number, string delimiter, string str)

Beskrivning

Extraherar ett element från en sträng av element läst från en file med fgets() funktionen. felement() kan användas istället för element() när den lästa strängen är längre än sträng-storleken 256. felement() kan hantera rader upp till 1023 tecken.

Argument

int	number	elementets nummer.
string	delimiter	avgränsnings tecken.

Exempel

```
string elem1;
int file;
string line;

file = fopen( "my_file.txt", "r");
while( fgets( line, file))
  elem1 = felement( 1, " ");
endwhile
```

30.8.3 fgets()

int fgets(string str, int file)

Beskrivning

Läser en rad från en angiven fil.
Returnerar noll vid filslut.

Argument

string	str	Läst rad. Returnerad.
int	file	Fil id returnerad av fopen.

Exempel

```
file = fopen("some_file.txt","r");
while( fgets( str, file))
  say( str);
endwhile
fclose( file);
```

30.8.4 file_search()

int file_search(string pattern, string found_file, int pass)

Beskrivning

Söker efter filer.
Ett sökmönster med wildcard kan anges för att söka på flera filer.
Söksekvensen är indelad i passen init, next och end. Vid första
anropet anges pass init (1). Vid sökning av flera filer med samma
mönster anges pass next (0). Sökningen avslutas med pass end (2).
Returnerar udda status om en fil har hittats, annars jämn status.

Argument

string	pattern	Namn på fil som ska sökas efter, kan innehålla wildcard ('*').
string	found_file	Hittad fil.
int	pass	Pass. Init (1), next (0) eller end (2).

Exempel

```
string pattern = "*.txt";
string found_file;
int sts;

sts = file_search(pattern, found_file, 1);
while (sts & 1)
  printf("Processing %s\n", found_file);
  ...
  sts = file_search(pattern, found_file, 0);
endwhile
file_search(pattern, found_file, 2);
```

30.8.5 fopen()

int fopen(string filespec, string mode)

Beskrivning

Öppnar en fil för att läsa eller skriva.
Returnerar en fil identitet, Om filen inte kunde öppnas, returneras noll.

Argument

string	filespec	Namn på filen.
string	mode	Access mod

Exempel

```
int infile;
int outfile;

infile = fopen("some_file.txt","r");
outfile = fopen("another_file.txt","w");
...
fclose( infile);
fclose( outfile);
```

30.8.6 fprintf()

int fprintf(int file, string format [, (godtycklig typ) arg1,
(godtycklig typ) arg2])

Beskrivning

Formaterad utskrift på fil. C-syntax. Format argument och inget, ett eller två värde argument.
Returnerarn antal utskrivna tecken.

Argument

int	file	Fil id returnerat av fopen.
string	format	Format.
godtycklig typ	arg1	Värde argument. Valfritt. Kan vara int, float eller string.
godtycklig typ	arg2	Värde argument. Valfritt. Kan vara int, float eller string.

Exempel

```
int outfile;
outfile = fopen( "my_file.txt", "w");
if (!outfile)
  exit();
fprintf( outfile, "Some text");
fprintf( outfile, "a = %d", a);
fclose( outfile);
```

30.8.7 fscanf()

int fscanf(int file, string format, (godtycklig typ) arg1)

Beskrivning

Formaterad läsning från fil. C-syntax.
Returnerar antal inlästa tecken.

Argument

int	file	Fil id.
string	format	Format.
godtycklig typ	arg1	Värde argument. Returnerat. Kan vara int, float eller string.

Exempel

```
int file;
int i;

file = fopen( "my_file.txt", "r");
if (file)
  fscanf( file, "%d", i);
  fclose( file);
endif
```

30.8.8 translate_filename()

string translate_filename(string fname)

Beskrivning

Byter ut omgivnings variabler i ett filnamn.

Argument

string	fname	Ett filnamn.

Returns

string	Sträng med utbytta omgivningsvariabler.

Exempel

```
string fname1 = "$pwrp_db/a.wb_load";
string fname2;
fname2 = translate_filename( fname1);
```

30.9 Sträng funktioner

Funktion	Beskrivning
edit	Ta bort överfödiga mellanslag och tab.
element	Extrahera ett element från en sträng.
extract	Extrahera en delsträng från en sträng.
sprintf	Formaterad utskrift till en sträng.
strchr	Returnera första förekomsten av ett tecken i en sträng.
strrchr	Returnera sista förekomsten av ett tecken i en sträng.
strlen	Beräkna längden på en sträng.
strstr	Returnera första förekomsten av en delsträng i en sträng.
tolower	Konvertera till små bokstäver.
toupper	Konveretra till stora bokstäver.

30.9.1 edit()

string edit(string str)

Beskrivning

Tar bort inledande och avslutande blanktecken och tabbar, och ersätter flera tabbar och blanktecken med ett blanktecken.
Returnerar den editerade stängen.

Argument

string	str	sträng som ska editeras.

Exempel

```
collapsed_str = edit(str);
```

30.9.2 element()

string element(int number, string delimiter, string str)

Beskrivning

Extraherar ett element från en sträng av element.
Returnerar det extraherade elementet.

Argument

int	number	elementets nummer.
string	delimiter	avgränsnings tecken.
string	str	sträng med element.

Exempel

```
string str = "mary, lisa, anna, john";
string elem1;
elem1 = elment( 1, ",", str);
```

30.9.3 extract()

string extract(int start, int length, string str)

Beskrivning

Extraherar de angivna tecknen från angiven sträng.
Returnerar de extraherade tecknen som en sträng.

Argument

int	start	start positionen för första tecknet. Första tecknet har position 1.
int	length	antalet tecken som ska extraheras.
string	str	sträng som tecknen ska extraheras från.

Exempel

```
extracted_str = extract( 5, 7, str);
```

30.9.4 sprintf()

int sprintf(string str, string format [, (arbitrary type) arg1, (arbitrary type) arg2])

Beskrivning

Formaterad utskrift. C-syntax. Format argument och inget, ett eller två värde argument.
Returnerar antal utskrivna tecken.

Argument

string	str	Sträng att skriva till.
string	format	Format.
godtycklig typ	arg1	Värde argument. Valfritt, Kan vara int, float eller string.
godtycklig typ	arg2	Värde argument. Valfritt. Kan vara int, float eller string.

Exempel

```
string str;
int items;

sprintf( str, "Number of items: %d", items);
```

30.9.5 strchr()

int strchr(string str, string c)

Beskrivning

Returnerar första förekomsten av ett tecken in en sträng.

Argument

string	str	Sträng att söka i.
string	c	Tecken att söka efter.

Returns

int	Index för första förekomsen av ett tecken. Första tecknet har index 1. Returnerar noll om tecknet inte hittas.

Exempel

```
string str = "index.html";
int idx;

idx = strchr( str, ".");
```

30.9.6 strrchr()

int strrchr(string str, string c)

Beskrivning

Returnerar sista förekomsten av ett tecken in en sträng.

Argument

string	str	Sträng att söka i.
string	c	Tecken att söka efter.

Returns

int	Index för den sista förekomsten av tecknet. Första tecknet har index 1. Returnerar noll om tecknet inte hittas.

Exempel

```
string str = "/usr/local/pwrrt";
int idx;

idx = strrchr( str, "/");
```

30.9.7 strlen()

int strlen(string str, string c)

Beskrivning

Beräknar längden av en sträng.

Argument

string	str	Sträng att beräkna längden för.

Returns

int	Strängens längd.

Exempel

```
string str = "/usr/local/pwrrt";
int len;

len = strlen( str);
```

30.9.8 strstr()

int strstr(string str, string substr)

Beskrivning

Returnar första förekomsten av en delsträng i en sträng.

Arguments

string	str	Sträng att söka i.
string	substr	Delsträng att söka efter.

Returns

int	Index för första förekomsten av delsträngen. Första tecknet har index 1. Returnerar noll om delsträngen inte hittas.

Exempel

```
string str = "index.html";
int idx;

idx = strstr( str, ".html");
```

30.9.9 tolower()

string tolower(string str)

Beskrivning

Konverterar en sträng till små bokstäver.

Argument

string	str	sträng som ska konverteras.

Returns

string	sträng med små bokstäver.

Exempel

```
string str1 = "Buster Wilson";
string str2;
str2 = tolower( str);
```

30.9.10 toupper()

string toupper(string str)

Beskrivning

Konverterar en sträng till stora bokstäver.

Argument

string	str	sträng som ska konverteras.

Returns

string	sträng med stora bokstäver.

Exempel

```
string str1 = "Buster Wilson";
string str2;
str2 = toupper( str);
```

30.10 System funktioner

Funktion	Beskrivning
exit	Avsluta scriptet.
get_namespace	Hämta aktuell namespace.
set_namespace	Sätt namespace för externa variabler.
system	Exekvera ett shell kommando.
terminate	Avsluta processen.
time	Hämta systemtiden.
tzset	Sätt tidszon.
verify	Skriv ut exekverade rader.

30.10.1 exit()

int exit()

Beskrivning

Avslutar exekveringen av en fil.

Exempel

```
exit();
```

30.10.2 get_namespace()

string get_namespace()

Beskrivning

Returnerar aktuell namespace.

Exempel

```
string current_namespace;
current_namespace = get_namespace();
```

30.10.3 set_namespace()

set_namespace(string namespace)

Beskrivning

Sätt namespace för externa variabler.
Maxstorleken på namespace är 31 tecken. Om längden på argumentet överstiger maxlängden används de 31 sista tecknen i strängen.

Argument

string	namespace	Ny namespace.

Exempel

```
set_namespace(p1);
```

30.10.4 system()

int system(string cmd)

Beskrivning

Exekvera ett shell kommando.

Argument

string	cmd	Shell kommando att exekvera.

Returns

int	Returvärdet är -1 vid fel eller annars returstatus för kommandot.

Exempel

```
string cmd;

cmd = "firefox http://www.proview.se";
system( cmd);
```

30.10.5 terminate()

int terminate()

Beskrivning

Avsluta processen.

30.10.6 time()

string time()

Beskrivning

Returnerar nuvarande tid i strängformat.

Exempel

```
string t;
t = time();
```

30.10.7 tzset()

string tzset(string timezone)

Beskrivning

Sätt tidszon.

Exempel

```
tzset("Europe/Stockholm");
```

30.10.8 verify()

int verify([int mode])

Beskrivning

Sätter eller visar verifikatione mod. Om verifiering är till, visas alla
exekverade rader på skärmen.
Returnerar nuvarande verifikations mod.

Argument

int	mode	verifikaion till (1) eller från (0). Valfri.

Exempel

```
verify(1);
```

30.11 Databas funktioner

Funktion	Beskrivning
GetAttribute()	Hämta värde för ett attribut.
GetChild()	Hämta barn till ett objekt.
GetParent()	Hämta föräder till ett objekt.
GetNextSibling()	Hämta syskon för ett objekt.
GetNextVolume()	Hämta nästa volym.
GetClassList()	Hämta första instansen av en klass.
GetNextObject()	Hämta nästa instans av en klass.
GetClassListAttrRef()	Hämta första instansen av en klass, attribut-object inkluderade.
GetNextAttrRef()	Hämta nästa instans av en klass, attribut-objekt inkluderade.
GetTemplateObject()	Hämta template objektet för en klass.
GetNextTemplateAttrRef()	Hämta nästa instance i ett template objekt.
GetObjectClass()	Hämta klassen för ett objekt.
GetNodeObject()	Hämta nod-objektet.
GetRootList()	Hämta första objektet i rot-listan.
GetVolumeClass()	Hämta klassen för en volym.
GetVolumeList()	Hämta första volymen.
SetAttribute()	Sätt värde på ett attribut.
CreateObject()	Skapa ett objekt.
RenameObject()	Ändra namn på ett objekt.
MoveObject()	Flytta ett objekt.
InLib()	Testa om ett object ligger under en $LibHier.
OpenPlcPgm()	Öppna ett PlcPgm.
ClosePlcPgm()	Stäng ett PlcPgm.
CreatePlcObject()	Skapa ett plc-objekt.
CreatePlcConnection()	Skapa en plc-koppling.
SetPlcObjectAttr()	Sätt värde på ett attribut i ett plc-objekt.
PlcConnect()	Koppla ett plc-objekt.

30.11.1 GetAttribute()

(variabel typ) GetAttribute(string name [, int status])

Beskrivning

Hämta värdet för angivet attribut. Typen av det returnerade värdet beror på typen av attributet. Attributet kommer att konverteras till int, float eller string.

Argument

string	name	namn på attributet som ska hämtas.
int	status	status för operationen. Returnerad. Om noll kunde inte attributet hämtas. Valfritt.

Exempel

```
int alarm;
int sts;

alarm = GetAttribute("Roller-Motor-Alarm.ActualValue");
on = GetAttribute("Roller-Motor-On.ActualValue", sts);
if ( !sts)
  say("Could not find motor on attribute!");
```

30.11.2 GetChild()

string GetChild(string name)

Beskrivning

Hämta första barnet till ett objekt. Nästföljande barn kan hämtas med GetNextSibling().
Returnerar namnet på barnet. Om det inte finns något barn, returneras en null-sträng.

Argument

string	name	objektets namn.

Exempel

```
string child;

child = GetChild("Roller-Motor");
```

30.11.3 GetParent()

string GetParent(string name)

Beskrivning

Hämta föräldern till ett objekt.
Returnerar förälderns namn. Om det inte finns någon förälder returneras en null-sträng.

Argument

string	name	objektets namn.

Exempel

```
string parent;

parent = GetChild("Roller-Motor");
```

30.11.4 GetNextSibling()

string GetNextSibling(string name)

Beskrivning

Hämtar nästa syskon till ett objekt.
Returnerar namnet på syskonet. Om det inte finns något nästa systkon returneras en null-sträng.

Argument

string	name	objektets namn.

Exempel

```
string name;
int not_first;

name = GetChild("Rt");
not_first = 0;
while ( name != "")
  if ( !not_first)
    create menu/title="The Rt objects"/text="'name'"/object="'name'"
  else
    add menu/text="'name'"/object="'name'"
  endif
  not_first = 1;
  name = GetNextSibling(nname);
endwhile
if ( !not_first )
  MessageError("No objects found");
```

30.11.5 GetNextVolume()

string GetNextVolume(string name)

Beskrivning

Hämta nästa volym. Den första volymen hämtas med GetVolumeList(). Returnerar namnet på volymen. Om det inte finns någon nästa volym, returneras en null-sträng.

Argument

string	name	volymens hamn.

30.11.6 GetClassList()

string GetClassList(string class)

Beskrivning

Hämta första objektet av angiven klass. Nästa objekt av klassen kan hämtas med GetNextObject().
Returnerar namnet på första objektet. Om det inte finns några instanser av klassen returneras en null-sträng.

Argument

string	name	namn på klassen.

Exempel

```
string name;

name = GetClassList("Dv");
```

30.11.7 GetNextObject()

string GetNextObject(string name)

Beskrivning

Hämta nästa objekt i klasslistan.
Returnerar namnet på objektet. Om det inte finns något nästa objekt returneras en null-sträng.

Argument

string	name	objektets namn.

Exempel

```
string name;

name = GetClassList("Di");
while ( name != "")
  printf("Di object found: %s", name);
  name = GetNextObject(name);
endwhile
```

30.11.8 GetClassListAttrRef()

string GetClassListAttrRef(string class)

Beskrivning

Hämta första objekt eller attributeobjekt av angiven klass. Nästa objekt eller attributobjekt av klassen kan hämtas med GetNextAttrRef(). Returnerar namnet på första objektet. Om det inte finns några instanser eller attributobjekt av klassen returneras en null-sträng.

Argument

string	name	namn på klassen.

Exempel

```
string name;

name = GetClassListAttrRef("Dv");
```

30.11.9 GetNextAttrRef()

string GetNextAttrRef(string class, string name)

Beskrivning

Hämta nästa objekt i klasslistan.
Returnerar namnet på objektet eller attributobjektet. Om det inte finns
något nästa objekt returneras en null-sträng.

Argument

string	class	namn på klassen.
string	name	namn på objektet eller attributobjektet.

Exempel

```
string name;

name = GetClassListAttrRef("Di");
while ( name != "")
  printf("Di object found: %s", name);
  name = GetNextAttrRef( "Di", name);
endwhile
```

30.11.10 GetTemplateObject()

string GetTemplateObject(string class)

Beskrivning

Hämta template objektet för en specifik klass.

Argument

string	name	namn på klassen.

Exempel

```
string name;

name = GetTemplateObject("Dv");
```

30.11.11 GetNextTemplateAttrRef()

string GetNextTemplateAttrRef(string class, string name)

Beskrivning

Hämta nästa attribute objekt av den angivna klassen i ett template objekt. Returnerar namnet på attribut objektet. Om det inte finns något nästa objekt returneras en null-sträng. Det första template objektet hämtas med GetTemplateObject().

Argument

string	class	namn på klassen.
string	name	namn på template objekt och template attribute objekt.

Exempel

```
string name;

name = GetTemplateObject("Di");
while ( name != "")
  printf("Di object found: %s", name);
  name = GetNextTemplateAttrRef( "Di", name);
endwhile
```

30.11.12 GetObjectClass()

string GetObjectClass(string name)

Beskrivning

Hämta klassen för ett objekt.
Returnerar klassens namn.

Argument

string	name	objektets namn.

Exempel

```
string class;

class = GetObjectClass("Motor-Enable");
```

30.11.13 GetNodeObject()

string GetNodeObject()

Beskrivning

Hämta nod objektet.
Returnerar namnet på nod objektet.

Exempel

```
string node;
node = GetNodeObject();
```

30.11.14 GetRootList()

string GetRootList()

Beskrivning

Hämta första objektet i rot listan.
Returnerar namnet på rotobjektet. Nästa objekt i rotlistan kan hämtas med GetNextSibling().

Exempel

```
string name;

name = GetRootList();
while( name != "")
  printf( "Root object found: %s", name);
  name = GetNextSibling(name);
endwhile
```

30.11.15 GetVolumeClass()

string GetVolumeClass(string name)

Beskrivning

Hämta klassen för en volym.
Returnerar klassnamnet.

Argument

string	name	volymens namn.

Exempel

```
string class;

class = GetVolumeClass("CVolVKVDKR");
```

30.11.16 GetVolumeList()

string GetVolumeList()

Beskrivning

Hämta första volymen i volymslistan.
Returnerar namnet på volymen. Nästa volym kan hämtas med
GetNextVolume().

Exempel

```
string name;

name = GetVolumeList();
while( name != "")
  printf( "Volume found: %s", name);
  name = GetNextVolume(name);
endwhile
```

30.11.17 SetAttribute()

int SetAttribute(string name, (godtycklig typ) value)

Beskrivning

Sätt värde på ett attribut.
Attributet specificeras med fullt objekts och attributnamn.
Returnerar operationens status.

Argument

string name attributets name.
godtycklig typ value attributets värde.

Exempel

```
SetAttribute( "Pump-V1-Switch.Description", "Valve switch open");
```

30.11.18 CreateObject()

int CreateObject(string name, string class, string destination, int destcode)

Beskrivning

Skapa ett objekt.
Returnerar status för operationen.

Argument

string	name	objektsnamn. Utan path.
string	class	objektets klass.
string	destination	destinationsobjekt. Förälder eller syskon till objektet.
int	destcode	destinationskod. 1 första barn, 2 sista barn, 3 efter, 4 före.

Exempel

```
CreateObject( "Temperature", "BaseTempSensor", "Pump-V1", 2);
```

30.11.19 RenameObject()

int RenameObject(string name, string newname)

Beskrivning

Ändra namn på ett objekt.
Returnerar status för operationen.

Argument

string	name	tidigare namn med path.
string	newname	nytt namn utan path.

Exempel

```
RenameObject( "H1-Zon1-Temp2", "PT2");
```

30.11.20 MoveObject()

int MoveObject(string name, string destination, int destcode)

Beskrivning

Flytta ett objekt.
Returnerar status för operationen.

Argument

string	name	namn med path.
string	destination	destinationsobjekt. Förälder eller syskon till objektet i den nya positionen.
int	destcode	destinationskod. 1 första barn, 2 sista barn, 3 efter, 4 före.

Exempel

```
MoveObject( "H1-Zon1-Temp2", "H1-Zon2", 1);
```

30.11.21 InLib()

int InLib(string name)

Beskrivning

Testa om ett objekt ligger under en $LibHier.
Returnerar 1 om objektet ligger under en $LibHier, annars 0.

Argument

string	name	objektnamn med path.

Returnerar

int	1 om objektet ligger under en $LibHier, annars 0.

Exempel

```
if ( !InLib( "H1-Motor"))
  ...
endif
```

30.11.22 OpenPlcPgm()

int OpenPlcPgm(string name)

Beskrivning

Öppna ett plcprogram och en editeringssession för plc.
Returnerar status för operationen.
Det får inte finnas några osparade operationer när den här funktionen anropas.
Endast ett program kan vara öppet åt gången.

Editeringssessionen ska stängas med ett anropt ill ClosePlcPgm():
I editeringssessionen kan endast plc-funktionerna CreatePlcObject(), CreatePlcConnection(), SetPlcObjectAttr() och PlcConnect() användas för att modifiera och skapa objekt. Access för den tidigare sessionen är tillfälligt satt till readonly när plc-editering är aktiv.

Argument

string	name	namn på ett PlcPgm objekt.

Exempel

```
OpenPlcPgm( "Pump-V1-Control");
...
ClosePlcPgm();
```

30.11.23 ClosePlcPgm()

int ClosePlcPgm()

Beskrivning

Stänger en editeringssession för plc

Exempel

```
OpenPlcPgm( "Pump-V1-Control");
...
ClosePlcPgm();
```

30.11.24 CreatePlcObject()

int CreatePlcObject(string name, string class, float x, float y [, string destination,
int inputmask, int outputmask, int invertmask])

Beskrivning

Skapar ett funktionsobjekt i en editeringssession för plc.
Funktionen kan endast användas i en editeringssession för plc, som har öppnats med ett anrop till OpenPlcPgm().

Objektet positioneras på koordinaterna x och y. Om destination, ett dokumentobjekt, har angivits, är kordinateran relativa till detta objekt, annars absoluta.
Om maskerna inte anges används defaultmasker.

Argument

string	name	Objektsnamn, utan path.
string	class	Klass på objektet.
float	x	x-koordinat.
float	y	y-koordinat.
string	destination	Valfritt. Ett dokumentobjekt. Om ett dokumentobjekt är angivet är koordinaterna relativa till detta objekt.
int	inputmask	Valfritt. Mask där bitarna indikerar synliga ingångspinnar i funktionsblocket.
int	outputmask	Valfritt. Mask där bitarna indikerar synliga utgångspinnar i funktionsblocket.
int	invmask	Valfritt. Mask där bitarna indikerar inverterade ingångspinnar.

Exempel

```
OpenPlcPgm( "Pump-V1-Control");
CreatePlcObject( "Document0", "Document", 1.2, 0.0);
CreatePlcObject( "And0", "And", 0.3, 0.1, "Document0", 15, 1, 3);
CreatePlcObject( "V1", "BaseCValve", 0.3, 0.4, "Document0");
ClosePlcPgm();
```

30.11.25 CreatePlcConnection()

int CreatePlcConnection(string source, string sourceattr, string dest, string destattr [,
int feedback])

Description

Skapar en plc-koppling mellan två funktionsobjekt.

Funktionen kan endast användas i en editeringssession för plc som startas med ett anrop till OpenPlcPgm().

In- och utgångspinnarna som kopplingen ska anslutas till, anges med attributnamnen för pinnarna. Om en sträckad feedback-koppling ska skapas, kan detta anges med feedback argumentet.

Argument

string	source	Objekt som kopplingen utgår ifrån. Namn utan path.
string	sourceattr	Attribut för den pinne som kopplingen utgår ifrån.
string	dest	Destinationsobjekt. Namn utan path.
string	destattr	Attribut för pinne på destinationsobjektet.
int	feedback	Valfritt. Om 1 skapas en sträckad feedback-koppling.

Exempel

```
OpenPlcPgm( "Pump-V1-Control");
...
CreatePlcObject( "And0", "And", 0.3, 0.1, "Document0", 15, 1, 3);
CreatePlcObject( "And1", "And", 0.6, 0.1, "Document0", 15, 1, 3);
CreatePlcConnection( "And0", "Status", "And1", "In1");
# Feedback connection
CreatePlcConnection( "And1", "Status", "And0", "In4", 1);
ClosePlcPgm();
```

30.11.26 PlcConnect()

int PlcConnect(string plcobject, string connectobject)

Beskrivning

Aktiverar 'connect'funktionen för plc, till exempel för att koppla Get eller Sto funktionsobjekt till signaler eller attribut i planthierarkin.

Funktionen kan endast användas i en editeringssession för plc som startas med ett anrop till OpenPlcPgm().

Argument

string	plcobject	Namn, utan path, på det funktionsobjekt som ska kopplas.
string	connectobject	Namn på objekt som ska kopplas till. Fullständigt namn med path ska anges.

Exempel

```
OpenPlcPgm( "Pump-V1-Control");
...
CreatePlcObject( "GetDv0", "GetDv", 0.3, 0.1, "Document0");
PlcConnect( "GetDv0", "Pump-V1-Active");
...
ClosePlcPgm();
```

30.11.27 SetPlcObjectAttr()

int SetPlcObjectAttr(string attribute, (arbitrary type) value)

Beskrivning

Sätt ett värde i ett plc-objekt.

Funktionen kan endast användas i en editeringssession för plc som startas med ett anrop till OpenPlcPgm().
OpenPlcPgm().

Argument

string	name	Namn på attributet. Utan path.
arbitrary type	value	värde som ska sättas.

Exempel

```
OpenPlcPgm( "Pump-V1-Control");
CreatePlcObject( "Document0", "Document", 1.3, 0.0);
SetPlcObjectAttr( "Document0.DocumentOrientation", 1);
SetPlcObjectAttr( "Document0.DocumentSize", 3);
...
ClosePlcPgm();
```

30.12 Diverse funktioner

Funktion	Beskrivning
GetProjectName()	Hämta projektnamnet.
CheckSystemGroup()	Kontrollera att en systemgrupp existerar.
CutObjectName()	Klipp av ett objektsnamn.
MessageError()	Skriv ett felmeddelande.
MessageInfo()	Skriv ett informationsmeddelande.
GetCurrentText()	
GetCurrentObject()	
GetCurrentVolume()	
IsW1()	
IsW2()	
EditMode()	
MessageDialog()	
ConfirmDialog()	
ContinueDialog()	
PromptDialog()	
OpenGraph()	
CloseGraph()	
SetSubwindow()	
GetIoDeviceData()	
SetIoDeviceData()	
GetVersion()	
get_pwr_config()	
get_node_name()	
getmsg()	
EVEN()	
ODD()	

30.12.1 GetProjectName()

string GetProjectName()

Beskrivning

Hämta projektnamnet.
Returnerar projektets namn.

Exempel

```
string name;

name = GetProjectName();
```

30.12.2 CheckSystemGroup()

int CheckSystemGroup()

Beskrivning

Kontrollera om en system grupp existerar.
Returnerar 1 om systemgruppen existerar, annars 0.

Exempel

```
if ( !CheckSystemGroup( "MyGroup"))
  return;
endif
```

30.12.3 CutObjectName()

string CutObjectName(string name, int segments)

Beskrivning

Klipp av det första leden i ett objektsnamn.
Returnerar de sista segementen av ett objektnamn. Antalet segment som ska återstå anges med 'segments' argumentet.

Argument

string	name	Objektets hierarkinamn.
int	segments	Antal namnled som ska återstå.

Exempel

```
string path_name;
string object_name;

path_name = GetChild("Rt-Motor");
object_name = CutObjectName( path_name, 1);
```

30.12.4 MessageError()

string MessageError(string message)

Beskrivning

Skriv ett felmeddelande på skärmen.

Exempel

```
MessageError("Something went wrong");
```

30.12.5 MessageInfo()

string MessageInfo(string message)

Beskrivning

Skriv ett informations meddelande på skärmen.

Exempel

```
MessageInfo("Everything is all right so far");
```

30.12.6 GetCurrentText()

string GetCurrentText()

Beskrivning

Hämta texen på utvalt menyalternativ.

Exempel

```
string text;

text = GetCurrentText();
```

30.12.7 GetCurrentObject()

string GetCurrentObject()

Beskrivning

Hämta objekt som är associerat med utvalt meny alternativ.
Om inte något objekt är associerat returneras en null-sträng.

Exempel

```
string object;

object = GetCurrentObject();
```

30.12.8 GetCurrentVolume()

string GetCurrentVolume()

Beskrivning

Hämta volymen för aktuell session.
Om ingen aktuell session finns, returneras in null-sträng.

Exempel

```
string current_volume;

current_volume = GetCurrentVolume();
set volume/volume=SomeOtherVolume
...
set volume/volume='current_volume'
```

30.12.9 IsW1()

int IsW1()

Beskrivning

Returnerar 1 om nuvarande fönster med imatnings fokus är anläggningshierarki fönstret. Annars returneras 0.

30.12.10 IsW2()

int IsW2()

Beskrivning

Returnerar 1 om nuvarande fönster med imatnings fokus är nodhierarki fönstret. Annars returneras 0.

30.12.11 EditMode()

int EditMode()

Beskrivning

Returnerar 1 om wtt är i editerings mod.
Annars returneras 0.

30.12.12 MessageDialog()

MessageDialog(string title, string text)

Beskrivning

Visar en dialogruta för meddelanden.

Argument

string	title	Titel.
string	text	Meddelande text.

Exempel

```
MessageDialog( "Message", "This is a message");
```

30.12.13 ConfirmDialog()

int ConfirmDialog(string title, string text [, int cancel])

Beskrivning

Visar en dialog ruta med konfirmering.
Returnerar 1 om yes-knappen har tryckts, 0 om no-knappen har trycks.
Om det tredje argumentet (cancel) har adderats, visas en cancel knapp
i dialogrutan. Om cancel-knappen har trycks, eller om dialogrutan
har stängs, sätts cancel argumentet till 1.

Argument

string	title	Titel.
string	text	Konfirmerings text.
int	cancel	Valfritt. En cancel knapp visas. Cancel sätts till 1 om cancel-knappen har trycks, eller om dialogrutan har stängts.

Exempel 1

```
if ( ! ConfirmDialog( "Confirm", "Do you really want to..."))
  printf( "Yes is pressed\n");
else
  printf( "No is pressed\n");
endif
```

Exempel 2

```
int cancel;
int sts;

sts =  ConfirmDialog( "Confirm", "Do you really want to...", cancel);
if ( cancel)
  printf("Cancel is pressed\n);
  exit();
endif

if ( sts)
  printf( "Yes is pressed\n");
else
  printf( "No is pressed\n");
endif
```

30.12.14 ContinueDialog()

ContinueDialog(string title, string text)

Beskrivning

Visar en dialogruta med knapparna 'Continue' och 'Quit'.
Returnerar 1 om continue har trycks, 0 om quit har trycks.

Argument

string	title	Titel.
string	text	Meddelande text.

Exempel

```
if ( ! ContinueDialog( "Message", "This script will...");
  exit();
endif
```

30.12.15 PromptDialog()

int PromptDialog(string title, string text, string value)

Beskrivning

Visar en dialogruta med en prompt som promptar för en inmagning.
Returnerar 1 om yes-knappen är tryckt, 0 om cancel-knappen har trycks,
eller om dialogrutan har stängs.

Argument

string	title	Titel.
string	text	Text.
string	value	Innehåller inmatat värde.

Exempel

```
string name;

if ( PromptDialog( "Name", "Enter name", name))
  printf( "Name : '%s'\n", name);
else
  printf( "Cancel...\n");
endif
```

30.12.16 OpenGraph()

int OpenGraph(string name, int modal)

Beskrivning

Öppnar en Ge graf.
Grafen kan öppnas som modal eller ej modal. Modal innebär att exekveringen av scriptet inte fortsätter förrän grafen har stängts.

Argument

string	name	Namn på grafen.
int	modal	Modal mod.

Exempel

```
OpenGraph( "pwr_wizard_frame", 0);
```

30.12.17 CloseGraph()

int CloseGraph(string name)

Beskrivning

Stänger en Ge graf.

Argument

string	name	Namn på grafen.

Exempel

```
CloseGraph( "pwr_wizard_frame");
```

30.12.18 SetSubwindow()

int SetSubwindow(string name, string windowname, string source, int modal)

Beskrivning

Öppnar en Ge graf i ett window objekt i en tidigare öppnad graf.
I modal mod, forsätter inte exekveringen förrän kommandot 'release subwindow'
exekveras av någon trycknapp i bilden.

Argument

string	name	Namn på huvud grafen.
string	windowname	Namn på window objektet i vilken den specificerad grafen ska visas.
string	source	Namn på grafen som ska visas i window objektet.
int	modal	Modal mod.

Exempel

```
SetSubwindow( "pwr_wizard_frame", "Window1", "MyGraph", 1);
```

30.12.19 GetIoDeviceData()

int GetIoDeviceData(string object, string parameter, string value)

Beskrivning

Hämta data från IO konfigurationen.
Returnerar status för operatione,. udda vid framgång, jämn vid fel.

Kommandot använder metoder och måste exekveras i pwrs. Det fungerar inte i pwrc.

De supportade parametrarna för Profinet enheter är

NetworkSettings-DeviceName
NetworkSettings-IP Address
NetworkSettings-Subnet Mask
NetworkSettings-MAC Address
NetworkSettings-SendClock
NetworkSettings-ReductionRatio
NetworkSettings-Phase
NetworkSettings-API

Argument

string	object	Namn på device objekt.
string	parameter	Parameter namn för data.
string	value	Returnerat parametervärde.

Exempel

```
sts = SetIoDeviceData( "Nodes-MyNode-PN-D1", "NetworkSettings-DeviceName", "ET200M-I
```

30.12.20 SetIoDeviceData()

int SetIoDeviceData(string object, string parameter, string value)

Beskrivning

Sätt data i IO konfigurationen.
Returnerar status för operationen, udda för framgång, jämn för fel.

Kommandot använder metoder och måste exekveras i pwrs. Det fungerar inte i pwrc.

De supportade parametrarna för Profinet enheter är

NetworkSettings-DeviceName
NetworkSettings-IP Address
NetworkSettings-Subnet Mask
NetworkSettings-MAC Address
NetworkSettings-SendClock
NetworkSettings-ReductionRatio
NetworkSettings-Phase
NetworkSettings-API

Argument

string	object	Namn på device objekt.
string	parameter	Parameter namn för data.
string	value	Värde att sätta.

Exempel

```
sts = SetIoDeviceData( "Nodes-MyNode-PN-D1", "NetworkSettings-DeviceName", "ET200M-I
```

30.12.21 GetVersion()

int GetVersion()

Beskrivning

Hämta ProviewR-versionen för aktuell utgåva.
Returnerar ett heltalsvärde som är 10000 * major + 100 * minor + release.
T ex V5.3.1 returnerar 50301.

Exempel

```
if ( GetVersion() > 50300)
  # Only for versions larger than V5.3.0
  ...
endif
```

30.12.22 get_pwr_config()

string get_pwr_config(string name)

Beskivning

Hämta värdet för en ProviewR konfigureringsvariabel.
Konfigureringsvariabler sätts i /etc/proview.cnf.
Returnerar värdet på konfigureringsvariablen.

Exempel

```
group = get_pwr_config( "defaultSystemGroup");
```

30.12.23 get_node_name()

string get_node_name()

Beskrivning

Hämta nodnamnet för aktuell nod.
Returnerar nodnamnet.

Exempel

```
name = get_node_name();
```

30.12.24 getmsg()

string getmsg(int status)

Beskrivning

Hämta texten för ett status värde.
Returnerar texten.

Exempel

```
msg = getmsg(sts);
```

30.12.25 EVEN()

int EVEN(int sts)

Beskrivning

Testa om ett heltal är jämnt.
Returnerar 1 om talet är jämnt, 0 om talet är udda.

Exempel

```
sts = SetAttribute( "Pump-V1-Switch.Description", "Valve switch open");
if ( EVEN(sts))
  printf("Couldn't set attribute\n");
endif
```

30.12.26 ODD()

int ODD(int sts)

Description

Testa om ett heltal är udda.
Returnerar 1 om heltalet är udda, 0 om det är jämnt.

Exempel

```
sts = SetAttribute( "Pump-V1-Switch.Description", "Valve switch open");
if ( ODD(sts))
  printf("Set operation successful\n");
endif
```

30.13 Wtt kommandon

Alla wtt kommandon finns tillgängliga i script koden. En wtt-kommando rad ska INTE avslutas med semikolon. Variabler kan substitueras i kommandot genom att omges av apostrofer.

Exempel

```
string name = "PUMP-VALVE-Open";
string value = "The valve is open";
set attribute/name='name'/attr="Description"/value='value'
```

Exempel

```
string name;
string parname;
int j;
int i;
for ( i = 0; i < 3; i++)
  parname = "vkv-test-obj" + (i+1);
  create obj/name='parname'
  for ( j = 0; j < 3; j++)
    name = parname + "-obj" + (j+1);
    create obj/name='name'
  endfor
endfor
```