



Joint PVSS & JCOP Framework Course

Course Manuscript

Issue: 2
Revision: 0

Reference: CERN-JCOP-2004-016
Created: 9 February 2004
Last modified: 22 March 2004

Prepared By: Sascha Marc Schmeling, CERN IT/CO

Abstract

The experiments participating in the Joint Controls Project (JCOP) and the IT/CO group at CERN have agreed to take over the training responsibilities for general PVSS and JCOP Framework training. For this, a combined course to teach the basics of the software packages PVSS II¹ and JCOP Framework has been set up. Subjects of this course are:

- overview of experiments' controls systems and their architecture in the LHC era,
- overview of the features and concepts of PVSS II,
- overview of the protocols used in LHC experiments,
- basic knowledge of PVSS usage and the ability to configure, run, and program small projects,
- basic configuration of the PVSS database, and
- basic usage of the JCOP Framework tools and devices.

This manuscript is meant as a reference for course participants. It is not intended to provide a full self-learning course. Several subjects covered in the course are not covered in this manuscript, because hand-outs or complete slide sets are available otherwise. These further references can be found on the IT/CO training web pages or directly at

<http://cern.ch/pvssjcopfw-training>

In case of questions or needed further explanations, please contact your trainer and/or the author of this manuscript. In case of questions about the supported products, please send your request to

ITControls.Support@cern.ch

In this manuscript, all tasks are performed on a Windows platform. PVSS and the JCOP Framework also work on Linux systems. The PVSS user interfaces on Linux however are not as sophisticated as the ones on Windows. JCOP Framework panels are usable on both platforms with more or less the same look. The graphical editor and parameterization modules of PVSS are different. As PVSS is platform independent, the author strongly recommends to develop at least your user interface on Windows, even if you choose to run your system on Linux.

¹ Prozeß-Visualisierungs- und SteuerungsSystem II. Generation from ETM AG, Austria

Table of Contents

1. INTRODUCTION TO EXPERIMENTS' CONTROLS SYSTEMS	1
1.1 JCOP FRAMEWORK OVERVIEW	2
1.2 HIERARCHIES AND FINITE STATE MACHINES	5
1.3 PVSS AND CONTROLS SYSTEMS SUPPORT AT CERN	8
2. FIRST STEPS IN PVSS II	9
2.1 BASIC TOOLS FOR PROJECT MANAGEMENT	9
2.1.1 <i>Project Administration Panel</i>	9
2.1.2 <i>The PVSS Console</i>	10
2.2 THE GRAPHICAL EDITOR AND THE PARAMETERIZATION MODULE	12
2.2.1 <i>Simple Example for a Datapoint Type – A Simple HV Channel</i>	12
3. FIRST STEPS WITH THE JCOP FRAMEWORK.....	14
3.1 HOW TO GET AND INSTALL THE JCOP FRAMEWORK.....	14
3.1.1 <i>Where to Get the JCOP Framework</i>	14
3.1.2 <i>A New PVSS II Project Using the JCOP Framework</i>	14
3.2 STARTING THE JCOP FRAMEWORK FOR THE FIRST TIME	17
3.2.1 <i>The Device Editor and Navigator Basics</i>	18
3.2.2 <i>The ELMB Framework Component</i>	26
3.2.3 <i>Using FSM with the Device Editor and Navigator</i>	27
3.2.4 <i>Trending in the JCOP Framework</i>	42
3.3 USER INTERFACES BASED ON THE JCOP FRAMEWORK.....	47
4. AN EXAMPLE CONTROLS SYSTEM	48

List of Figures

Figure 1 Controls technologies in the LHC era	1
Figure 2 DCS Development Process	2
Figure 3 JCOP Framework Overview	2
Figure 4 Overview of FW Provided Components.....	3
Figure 5 : JCOP Supported Communications Mechanisms and their Possible Use within the Control System	4
Figure 6 A Simple Control System Modelled using FSMs.....	5
Figure 7 Example of states and commands for a control unit	5
Figure 8 Example of states and commands for a device unit.....	6
Figure 9 Simple Partitioning Example	6
Figure 10 AWG Defined Partitioning Modes	7
Figure 11 Control Unit Functionality	7
Figure 12 Device Unit Functionality	8
Figure 13 Screenshots from the IT/CO support pages	8
Figure 14 The new project administration panel	9
Figure 15 Possible actions from the project administration panel.....	9
Figure 16 Steps to create a standard project.....	10
Figure 17 The PVSS Console for an example project.....	10
Figure 18 Console project actions.....	11
Figure 19 Console manager actions	11
Figure 20 Manager properties panel	11
Figure 21 Simple Model of a HV channel.....	12
Figure 22 Steps to create a datapoint type	13
Figure 23 Main part of the JCOP Framework download pages.	14
Figure 24 Extracting the JCOP Framework Installation Tool to the PVSS project's folder	15
Figure 25 Run the installation tool from the graphical editor.....	15
Figure 26 Setting the installation directory for the JCOP Framework.	15
Figure 27 An example for available components for the JCOP Framework.	16
Figure 28 Overview of components chosen for installation.....	17
Figure 29 Adding an OPC client driver with number 6 and a control manager for the framework scripts with number 2. Please note that number 6 is obligatory for CAEN devices in the JCOP Framework in order to use the automatic setup, whereas number 2 for the control manager is a choice up to you.....	17
Figure 30 The JCOP Framework's DEN at first start in a new project, in navigator mode (left) and editor mode (right).	18
Figure 31 Managing devices in the DEN.....	19

Figure 32 Creating an analog input device and choosing a meaningful name.....	19
Figure 33 The DEN after creation of the analog input device.	20
Figure 34 Creating a CAEN crate device	20
Figure 35 Inserting a board and its channels into a CAEN SY1527 device	21
Figure 36 Operation panel for an analog input device	21
Figure 37 Debug mode for an analog input device	22
Figure 38 Operation panels for a CAEN crate, board, and channel.....	22
Figure 39 Device Action Panel.....	23
Figure 40 High Voltage Actions panel.....	23
Figure 41 Expert configuration of a framework device	23
Figure 42 Hardware and logical view example	24
Figure 43 Creation of nodes in the logical view	24
Figure 44 Adding hardware devices to the logical view and renaming them	25
Figure 45 Creating a logical object.....	27
Figure 46 Configuration of a logical object type (standard left, simple configuration utility right).	28
Figure 47 Steps to create a device unit type. Note that one should check “As Device Base Type”.	29
Figure 48 Configuration of a device unit type (standard left, simple configuration utility right).	29
Figure 49 Configuration possibilities for device unit types.	30
Figure 50 Simple state configuration panel.....	30
Figure 51 Basic creation steps for the complex device unit type	31
Figure 52 Basic state setup for a voltage channel	31
Figure 53 Steps to include a new state	32
Figure 54 Steps to configure the VoltageLayer.....	36
Figure 55 Generic control system architecture (FSM view)	37
Figure 56 Steps to create a root node.....	37
Figure 57 Adding nodes to a hierarchy	38
Figure 58 Stopping and starting the hierarchy	38
Figure 59 Viewing a hierarchy.....	39
Figure 60 A taken control unit and the corresponding ownership options	39
Figure 61 Simple operations with the DCSNode-type control units	39
Figure 62 Steps to include the voltage layer for the vertex sub-detector	40
Figure 63 Steps to include voltage channels	40
Figure 64 Branching through a running hierarchy.....	41
Figure 65 Sketch on pages and plots.....	42
Figure 66 Main trending tool panel.....	42

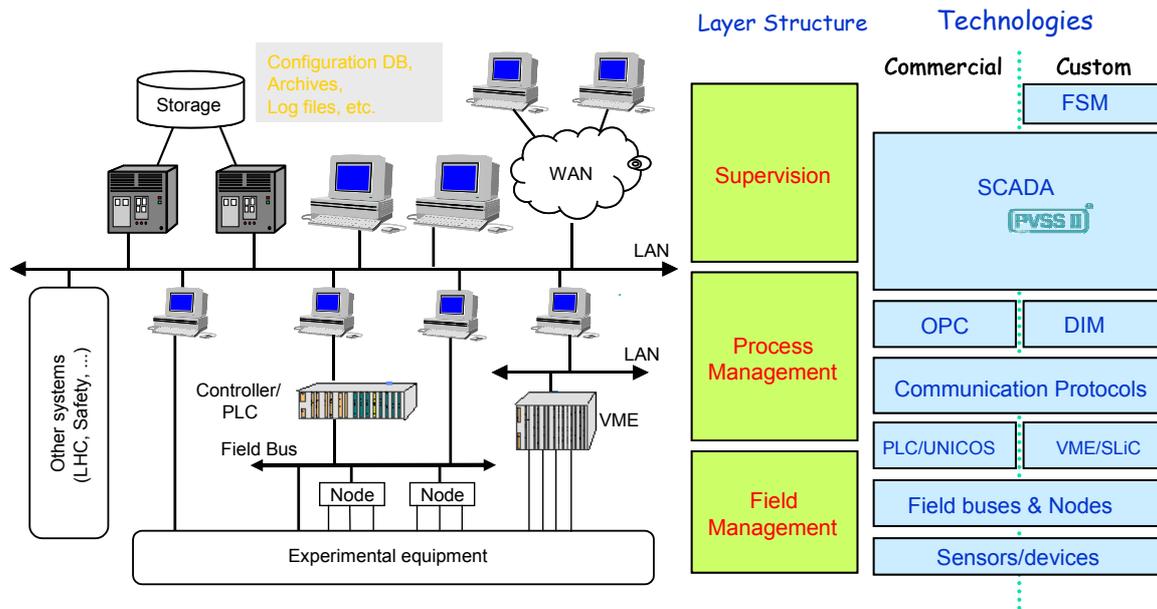
Figure 67 Handling of pages, plots, and trending tree nodes	43
Figure 68 Basic steps to create a new plot	43
Figure 69 Basic steps to create a new page	44
Figure 70 A single plot and some of the options	45
Figure 71 A page made of two plots and some page global options.....	45
Figure 72 Steps to create a node and arrange a plot inside	46

List of Tables

Table 1 States, actions, and WHEN clauses for the example <i>DCSNode</i> . Note that the order of the WHEN clauses is significant!.....	28
Table 2 States and Conditions for the MixedWater Device Unit Type	30
Table 3 States and Commands for a HV Channel	30
Table 4 Standard script for a device unit type of a CAEN channel.	32
Table 5 States and actions of a VoltageLayer	33

1. Introduction to Experiments' Controls Systems

The control system of each experiment will be an integration of multiple developments made by the various sub-detector teams as well as developments made by the experiment central teams. In general, each sub-detector will have a number of different systems to control and these may be developed by separate teams. In addition, a typical control system application will be built using a number of technologies and different layers as depicted in Figure 1.



Based on an original idea from LHCb

Figure 1 Controls technologies in the LHC era

Furthermore, a typical controls application will consist of a number of components which are used in many sub-detectors as well as some specific to that particular application. It is the aim of JCOP to provide and support such common components that are used in many sub-detectors.

As shown in Figure 2 the development of an experiment control system is seen as a multi-layered process. An Architecture Working Group (AWG) was established as part of JCOP to define the design [1] for the JCOP Framework (JCOPFw), which is intended to cover all the common needs of all four experiments. The framework is being implemented by the JCOPFw Team using the standard selected tools, e.g. PVSS and SMI++. PVSS is a commercial Supervisory Control And Data Acquisition (SCADA) tool from ETM and is the CERN-recommended SCADA solution. SMI++ is the Finite State Machine (FSM) toolkit that was developed for the DELPHI experiment and has been used in other experiments such as BaBar. This has been closely integrated with PVSS to benefit from the strengths of both products.

The output of the framework team will be an experiment independent framework supporting the common needs of all four experiments. However, individual experiments may have additional common requirements across all their sub-detectors, which should logically be implemented centrally, and therefore one can imagine a customization of the *Control System Framework* for each experiment. This *Experiment Framework* would then be used as basis for the development of the final DCS application. This development approach avoids unnecessary duplication and reduces the development effort required by the experiment sub-detector teams.

Hence, before commencing development each sub-detector team should discuss with its central DCS team regarding how to develop its controls application to fit into the overall strategy for the experiment. This may impose constraints on the design of the control system application as well as the interfaces it would have to abide by. Furthermore, the central DCS team may be able to offer common experiment components and support in addition to those provided by

the FW Team.

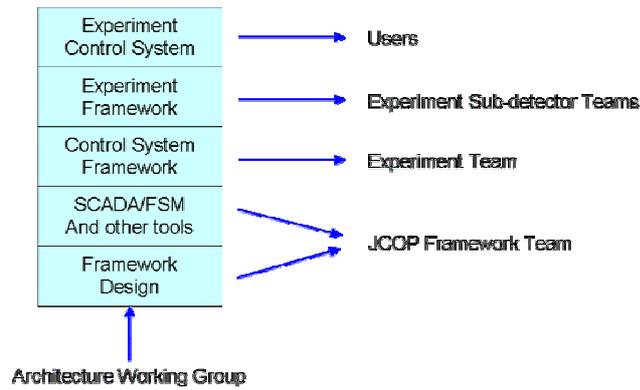


Figure 2 DCS Development Process

1.1 JCOP Framework Overview

As discussed above, JCOP has selected and is supporting a number of tools. In order to provide these in a directly usable fashion for the sub-detector teams to build their part of the DCS, and to reduce the overall development effort, the JCOPFW has been developed. This framework is an integrated set of guidelines [2] and tools that ease the development of control system applications. The framework is intended to include, as far as possible, all templates, standard elements and functions required to achieve a homogeneous control system and to reduce the development effort as much as possible. Furthermore, the framework is intended to hide the complexities of the underlying tools to reduce the knowledge required by a typical developer of the controls application. The JCOPFW is based on design decisions resulting from the work of the AWG.

Figure 3 gives an overview of the JCOPFW. As can be seen, the framework in principle covers all levels down to the connection to the hardware. However, as there is only agreement on the use of certain hardware elements and front-ends, the majority of the framework is provided at the supervisory level. Nevertheless, connection of other front-ends, and integration with the framework, is possible via one of a number of communications interfaces (OPC, PVSS II Communications, DIM or DIP).

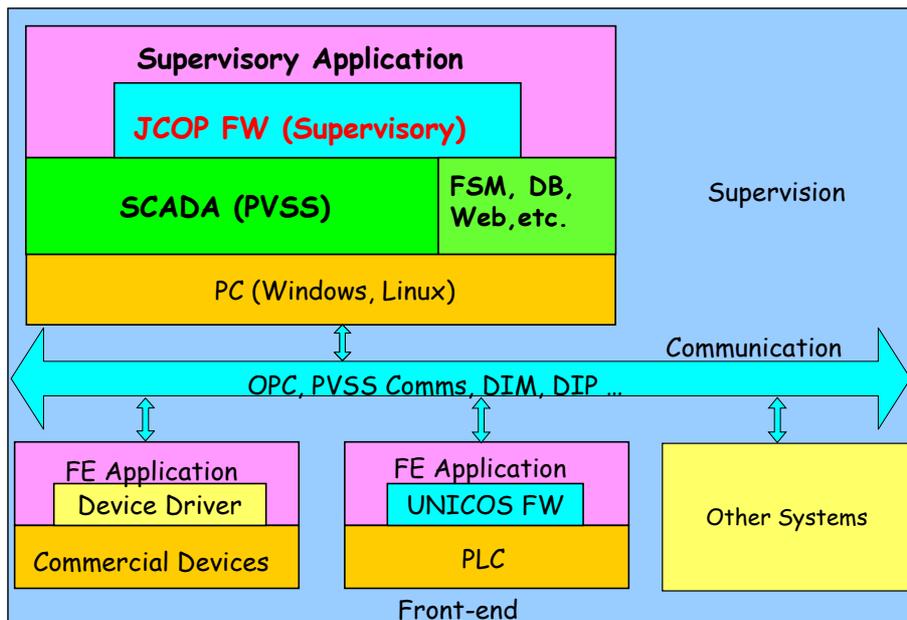


Figure 3 JCOP Framework Overview

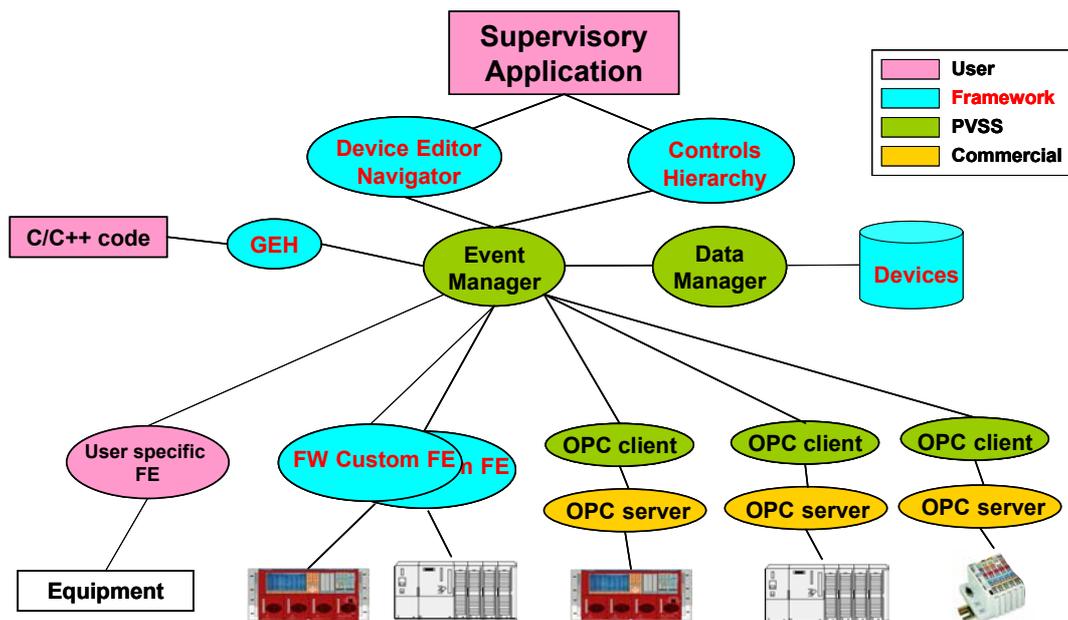


Figure 4 Overview of FW Provided Components

Figure 4 gives an overview of the various types of framework components. The colour coding is the same as for the previous Figure. In this way one can see those elements that are provided by the underlying tools, those which are added by the framework and those which the control system developer would need to add himself. The following gives an overview of the components (devices and tools) that are provided with the JCOPFw and those which are in development:

- Generic Analog-Digital devices
 - Analog Input/Output
 - Digital Input/Output
 - Process Monitor
 - Connection possible via OPC or DIM
- CAEN power supplies
 - OPC server available from the company
 - Crates SY127, SY403, SY527, SY1527 supported
 - Plans to include SY2527 and the EASY system, as well as two specialized cards for control of Wiener PL500 and for CMS MDT
- Wiener Power Supplies, Wiener Fan Trays and Wiener VME Crates
 - OPC server being developed by the company
 - CAN interface. One board currently supported (Kvaser)
 - Use of the OPC server with the Wiener Power Supplies is possible but must be done with care as there are some commands available that could potentially damage the equipment. For further information see:
<http://cern.ch/itcofe/Services/OPC/RecommendedTools/Servers/WIENER/welcome.html>
- ELMB
 - OPC server developed by ATLAS
 - CAN interface (CANOpen). One board currently supported (Kvaser)
- ISEG power supplies (prototype version exists but this is not yet in the official JCOPFw release)
 - OPC server being developed by the company
 - CAN interface. One board currently supported (Peak)
- PS and SPS machine data server (currently being updated for PVSS 3.0)
 - Common server provided for all experiments
 - SPS needs customization for each beam line
- Logical Node/View
 - Composite device
 - A means to build hierarchies of devices

- Device Editor/Navigator (DEN)
 - Main interface to the Framework
 - System management: installation, login, etc
 - Configuration and operation of devices
 - Provides three views of the system; Hardware, Logical and FSM
- Controls Hierarchy
 - High level view of experiment
 - Includes FSMs
 - Integrated in the DEN
- Trending Tool
 - Simplify & extend PVSS trends (templates, tree, etc)
 - Configuration database (prototype version available)
 - Allows to store and retrieve configuration information using an Oracle database
 - Allows to store and retrieve dynamic settings (recipes) using an Oracle database
- Device Support Extension
 - Template to incorporate new devices
- Generic External Handler (currently being updated for PVSS 3.0)
 - To incorporate C++ code (compiled functions) in panels and/or Ctrl managers
 - Easier to use than standard PVSS C++ interface
- Component Installation Tool
 - Tool for selectively installing framework components into a project
- Access Control (in development)
 - Customisation and extension of the PVSS access control mechanisms
- Tree Browser
 - Tree widget for Windows and Linux
- Rack Monitoring and Control (in development)
 - Application for configuring and operating LHC racks
- Integration of XML Parser in Ctrl (in development)
 - Possibility to parse XML files from within a Ctrl script

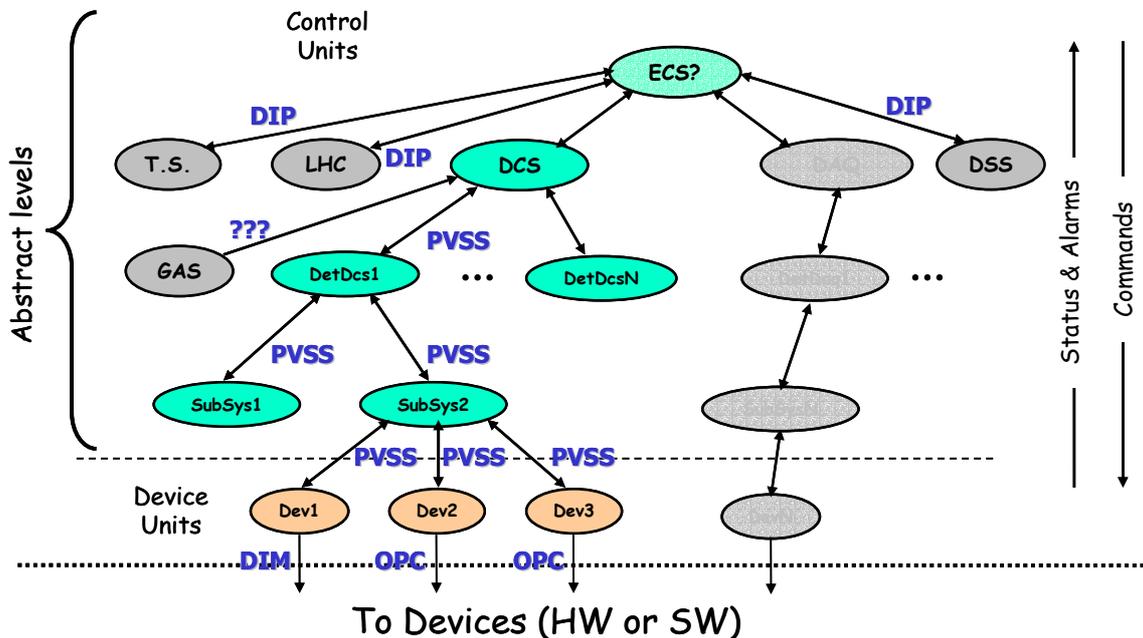


Figure 5 : JCOP Supported Communications Mechanisms and their Possible Use within the Control System

Figure 5 above identifies the three JCOP-supported communications mechanism (OPC, DIM and DIP) and an example of their possible use in the control system. Each of the three may also be used for different purposes. However, it is suggested to contact the experiment central team or IT-CO for advice on which protocol to use for which purpose.

1.2 Hierarchies and Finite State Machines

The AWG defined a method for modelling the control system as a hierarchy of finite state machines. In this model there is a state/command interface between a parent and its children. Commands are passed from a parent to its children and the states of the children are passed to the parent. The parent derives its state from those of its children. In this model, alarms are also propagated upwards through the hierarchy.

The AWG additionally defined two types of objects in this FSM hierarchy. These are Control Units (CUs) and Device Units (DUs). A simple example control system based on this modelling is shown in Figure 6.

DUs typically model a specific piece of equipment, e.g. motor, pump, HV channels, etc., and are capable of monitoring and controlling the equipment to which they correspond. CUs can model and control the sub-tree below them and typically model abstract or logical items, e.g. a subsystem or a sub-detector. DUs derive their states from hardware values which are read by PVSS and convert commands received from their parent to output parameters to the hardware. The DUs are thus modelled using PVSS scripting whereas the CUs, which derive their states from the states of their children, and possibly other objects, are modelled using a FSM toolkit. As stated above, the FSM toolkit chosen by JCOP is SMI++ and this has been fully integrated with PVSS within the framework.

As well as the differences described above, there is another major difference between a DU and a CU. The CU also supports two other concepts defined by the AWG, namely partitioning and ownership. These concepts will be elaborated later in this section.

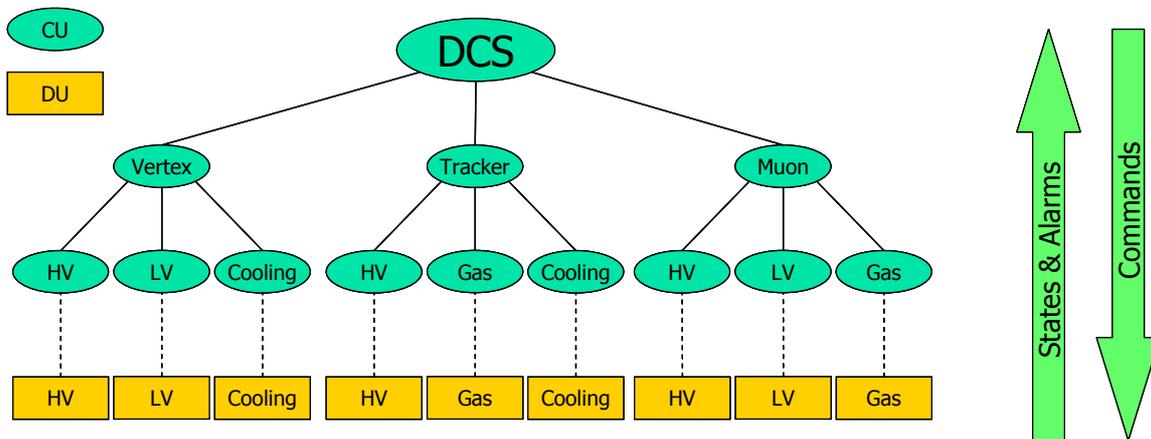


Figure 6 A Simple Control System Modelled using FSMs

An FSM may be thought of as a generic, data-driven, mechanism for modelling the functionality of a piece of equipment or a subsystem. The item to be modelled is thought of as having a set of stable, or quasi-stable, states. It can move between these states by making transitions, and transitions are triggered either by commands or state changes of a dependent FSM. In general, transitions are only possible between one state and a subset of the other states.

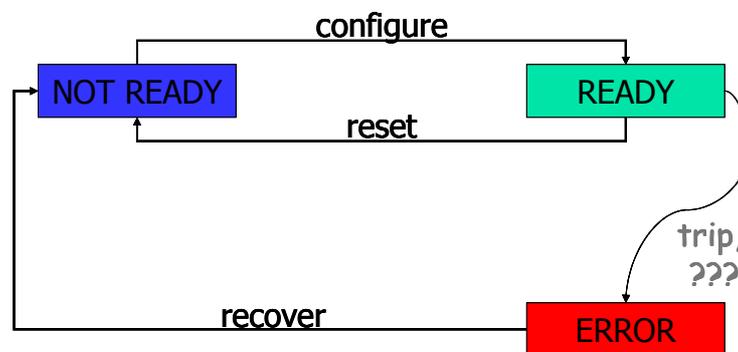


Figure 7 Example of states and commands for a control unit

Figure 7 shows a simple FSM for a CU. This type of diagram is also known as a State Transition Diagram (STD) and shows the possible states (two normal states and an error state) and the associated allowable transitions. When a

command is sent to Configure the CU this will imply that the CU sends corresponding commands to its children. When these children have confirmed that they are in the appropriate state the CU will move into the state READY. Figure 8 shows a simple STD for a DU. In this case, when the DU receives the command Switch On, it will generate an output signal to the equipment it is modelling to switch it on. When it receives input parameters to confirm that the equipment has been switched on it will move to the On state. This information will then be propagated to its parent CU to be taken into account for calculating its own state.

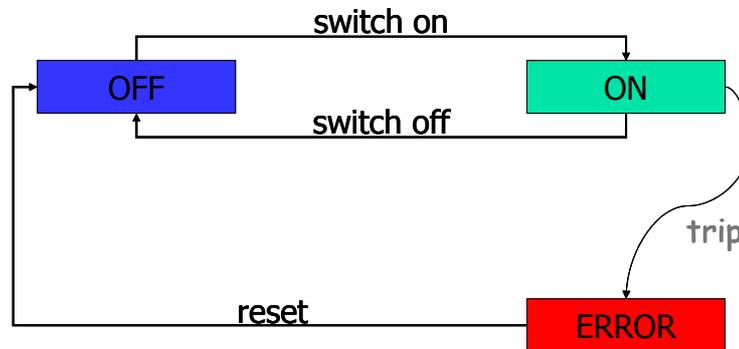


Figure 8 Example of states and commands for a device unit

During physics data taking the control system will generally be operated by the ‘shifter’ as an integrated system. Commands will be given to the top node and these will propagate down to the lower nodes as required based on the programming of the FSMs. However, there will be cases where one or more subsets of the overall control system need to be operated in a standalone manner and maybe even in parallel one to another. For this reason a number of partitioning modes were defined by the AWG.

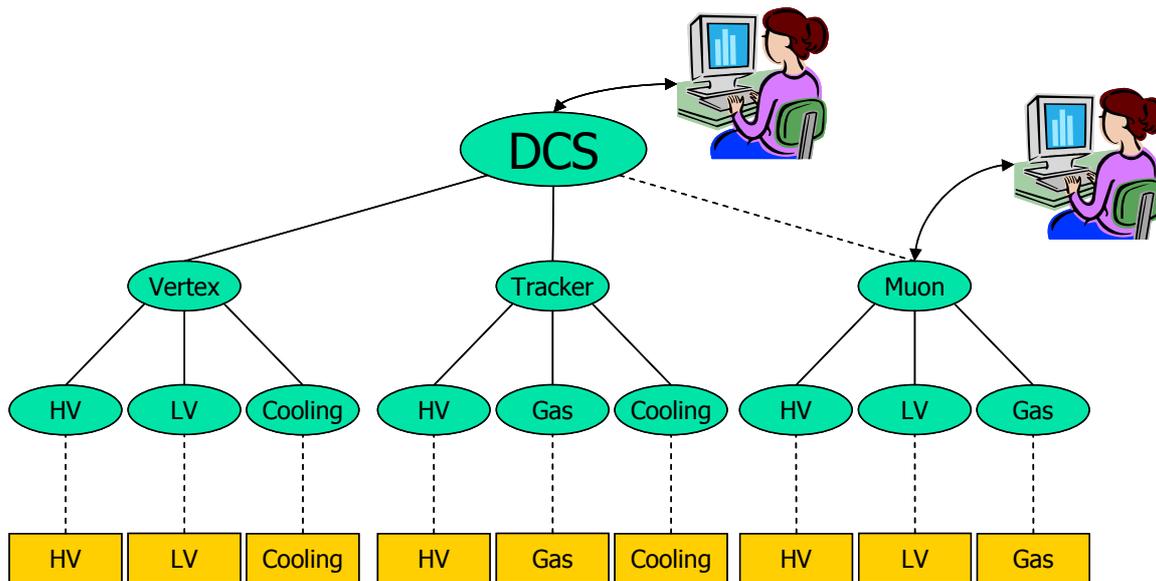


Figure 9 Simple Partitioning Example

Figure 9 can be used to explain a simple case of partitioning. In normal physics data taking the DCS will be operated as an integrated system of three detectors (Vertex, Tracker and Muon) each with three subsystems (HV, LV and Cooling) by the ‘shifter’ connecting his/her UIM to the DCS node. Another user connecting to the Muon detector node will be able to monitor the state of this detector and the corresponding three subsystems but will not be able to send any commands. One could imagine that after the run has finished that the Muon detector expert wishes to perform a calibration run without being affected by the operation of the rest of the DCS and without impacting on this either. Hence, it should be Excluded from the rest of the system, i.e. it will no-longer receive commands from the DCS node and its state will no longer be taken into account by the DCS node. After the calibration run the Muon detector node, and its sub-tree, would then be returned to the overall control system, i.e. it would be Included back into the DCS.

The above example gives the most typical cases of partitioning modes, i.e. Included and Excluded. However, the discussions in the AWG highlighted the need for more flexibility and hence two further partitioning modes were

defined, Manual and Ignored as can be seen in Figure 10. In the Manual mode a node is operated locally. Hence, it does not receive the commands from its parent, but its state is taken into account by its parent. This mode can be used to make sure the system will not send commands to a component while an expert is working on it. Since the component's state is still being taken into account, as soon as the component is fixed the operations will proceed. In the Ignored mode the component can be ignored by its parent, meaning that its state is not taken into account by the parent but it still receives commands. This mode can be useful if a component is reporting the wrong state or if it is only partially faulty and the operator wants to proceed nevertheless.

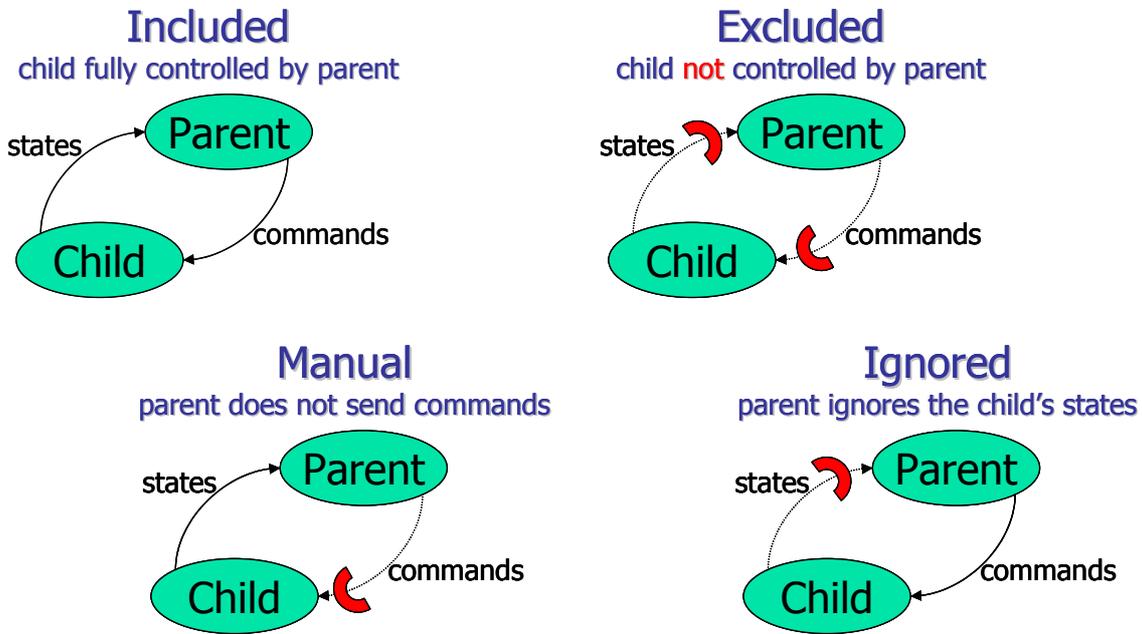


Figure 10 AWG Defined Partitioning Modes

Figure 11 and Figure 12 give an overview of the CU and DU functionality and implementation. Both provide the possibility to handle alarms and to log/archive data. The DU includes a device driver for communicating to the device it models. This would typically be a software device, but could also be a software process. Finally, it includes the FSM (command/state) interface to its parent. The CU by default includes the ownership and partitioning FSM. This FSM behaviour is common to all CUs and implements the ownership and partitioning behaviour defined by the AWG. The user is not required to define this behaviour, but rather it is inherited by default. However in addition, each CU includes an additional FSM for modelling the specific behaviour of that particular device. This behaviour must be defined by the user using the tools provided based on the State Management Language (SML).

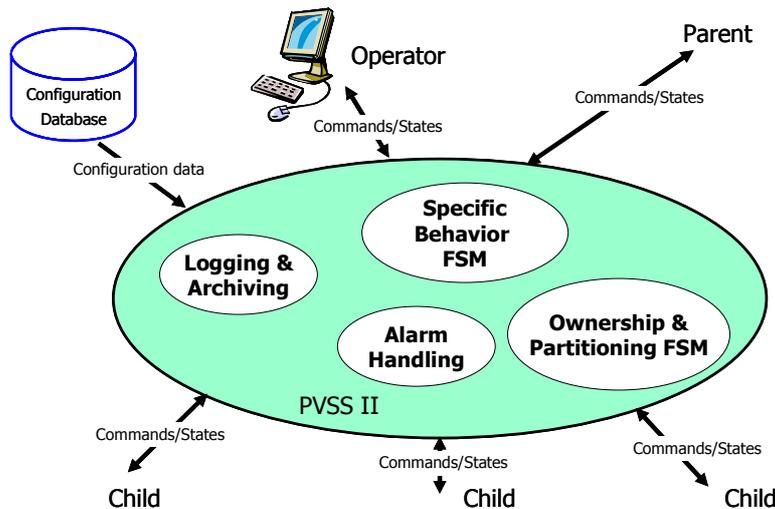


Figure 11 Control Unit Functionality

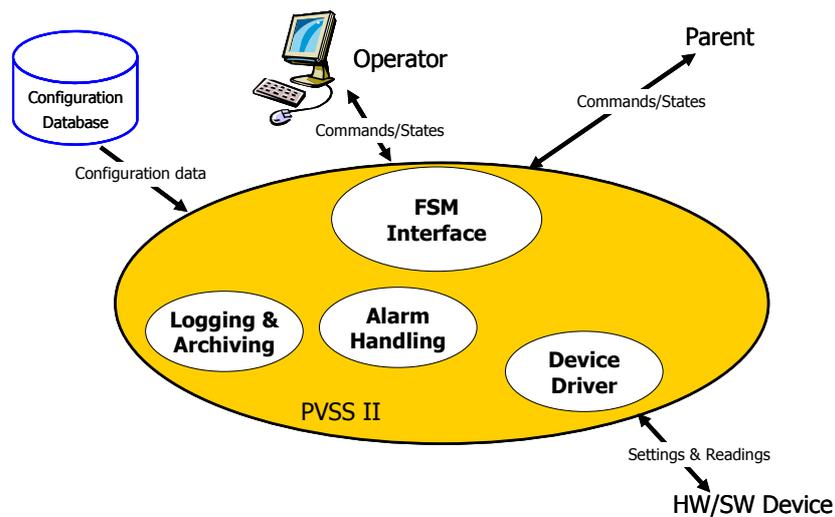


Figure 12 Device Unit Functionality

1.3 PVSS and Controls Systems Support at CERN

Support for experiment controls systems at CERN is organised in several levels. Each LHC experiment and every sector at CERN has a central controls systems team, that provides first level support and consultancy in terms of design to all users from that experiment.

On top of that, the controls systems support group IT/CO provides a support line reachable via the central eMail address

ITControls.Support@CERN.CH

Here, all PVSS and directly JCOP Framework related cases are handled. Furthermore, the front-end systems support is also reachable here.

In order to give advance support, the website of the IT/CO group provides a large variety of supporting information on controls systems at CERN:

<http://cern.ch/itco>

Where do I go from here?		Useful Links
If you are getting started and need PVSS	Getting PVSS	DSS
If you want an introduction to PVSS	PVSS Introduction	Fieldbus
If you need a PVSS course	Training	Framework
If you need a Framework course	SCADA Lab	JCOP
If you need a PVSS hands-on	FAQ	LHC GCS
If you need a Framework hands-on	Known Problems	OPC
	Development Tips	PLC
	Open Issues	PVSS Service
	Help Roadmap & Feedback	

Figure 13 Screenshots from the IT/CO support pages

In addition to that, a newsgroup called “cern.pvss” exists, where users can discuss encountered problems and their solutions. Note that this mailing list is not monitored continuously. Do not send support requests to it.

2. First Steps in PVSS II

This part of the manuscript is meant to give some examples on how to use PVSS in the area of high energy physics experiments, but does not detail the basic usage or concepts of PVSS itself. The course participants also receive the handout of ETM's basic PVSS course [1], where some more operations and especially the basic concepts of PVSS are explained.

2.1 Basic Tools for Project Management

Since version 3.0 of PVSS, there are three main tools for project management, the Project Administration panel, the Console (now also for Linux), and the Log Viewer.

2.1.1. Project Administration Panel

This new panel lets the user administer his projects, decoupling these tasks from the direct project running, which is controlled via the console. This and the inclusion of wizards makes many tasks easier and more intuitive. Figure 14 shows the panel on a PC with many registered projects.

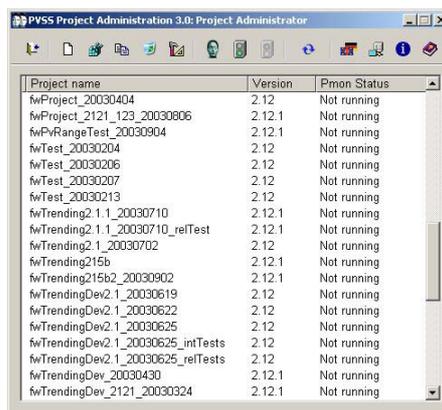


Figure 14 The new project administration panel

All main functions can be used from the toolbar at the top of the panel (see Figure 15). From left to right, the following actions can be performed:

- leave the project administration panel,
- create a new project,
- register a project (a project, even accessible file system-wise, needs to be registered in the PC to be able to be run),
- copy [and register] a project,
- delete a project,
- modify a project's properties,
- open the console for a specified project,
- start and stop a specified project,
- refresh the project list (manually or automatically),
- switch between the defined languages of the project,
- get the machine's hardware code (do not use this panel now for CERN applications), and
- show an about and the help pages.



Figure 15 Possible actions from the project administration panel

2.1.1.1. Creating a New Project

This is now aided by a wizard that asks the user different questions on the project to be created. Figure 16 shows the steps to create a standard project. Note that creating a new project takes time, as PVSS sets up databases and file structures from scratch.

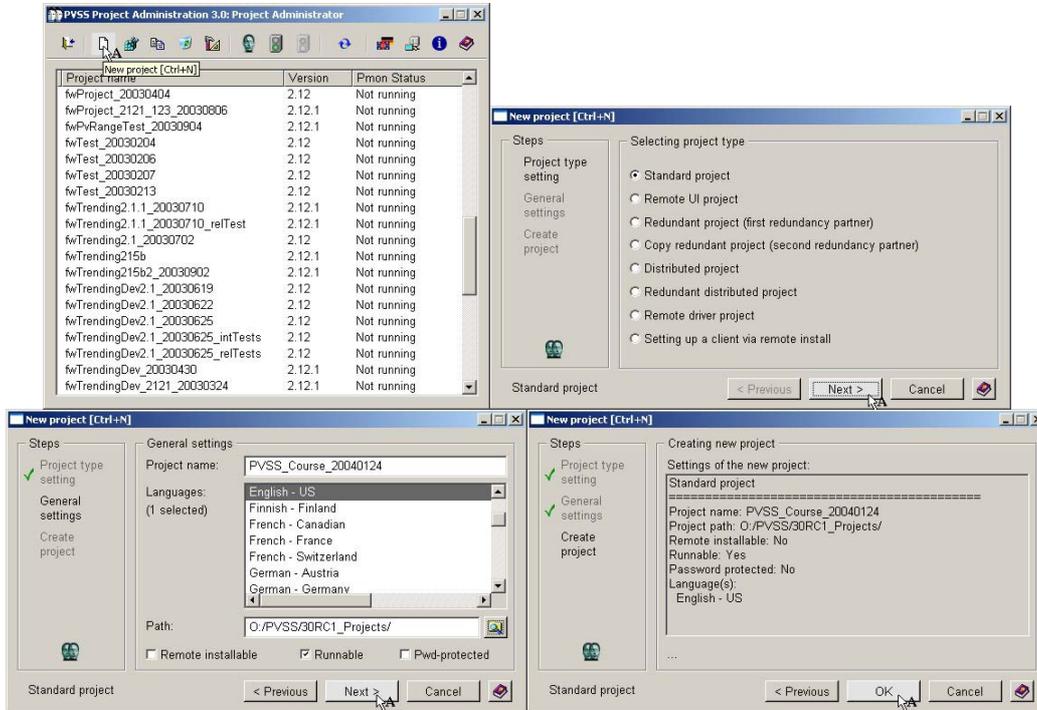


Figure 16 Steps to create a standard project

In the same way, also distributed or redundant projects can be built.

2.1.2. The PVSS Console

The console (see Figure 17) is now also available in Linux and facilitates project handling very much there. Essentially all tasks known from the Windows console have been integrated inside this panel, except the management tasks that are now in the project administration panel.

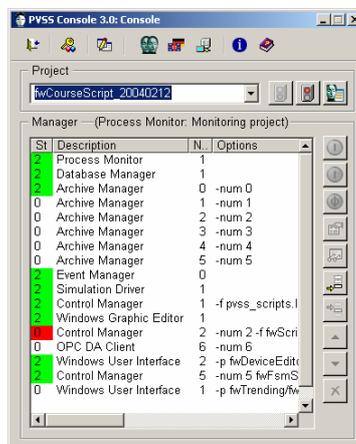


Figure 17 The PVSS Console for an example project

Figure 18 shows the toolbar buttons for project related actions, whereas Figure 19 shows the actions related to single

managers. A project as a whole can be chosen, locked, the language set, started and stopped, and its config file edited. Furthermore, one can open the project administration panel and the log viewer.

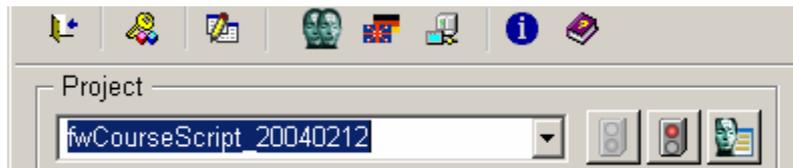


Figure 18 Console project actions

There are several actions related to single managers (see Figure 19):

- starting,
- stopping and killing,
- editing the manager's properties (see also Figure 20), and the activation of debug flags,
- inserting, ordering, and deleting.

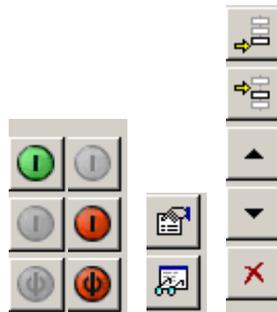


Figure 19 Console manager actions

The options for managers are the following (see Figure 20):

- command line options,
- start mode: **always** (according to the restart and reset counter parameters), **manual** (only started once manually and not restarted in case of stopping or killing), and **once** (once at each start of the complete project and not restarted),
- kill timer (seconds after stopping at which a kill signal is sent automatically), and
- not killing the manager in case of a project restart.

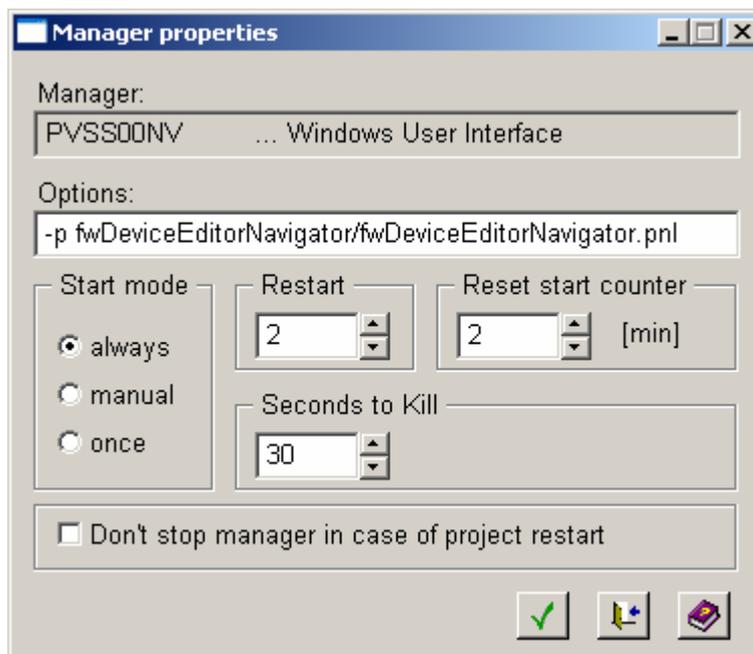


Figure 20 Manager properties panel

2.2 The Graphical Editor and the Parameterization Module

The functionalities of those tools is explained in [1], so in the following, only the major steps in order to perform the example are shown.

2.2.1. Simple Example for a Datapoint Type – A Simple HV Channel

The example used by ETM in order to motivate the approach of datapoint types and their instances is a valve. In the following, the example will be a simple HV channel model (see Figure 21).

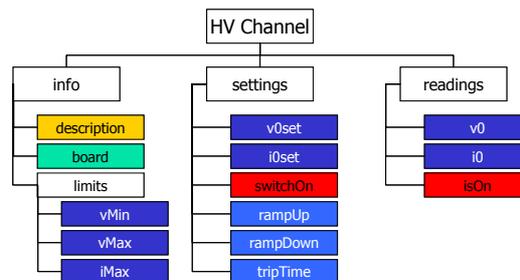


Figure 21 Simple Model of a HV channel

In this model, items are structured into information items, settings, and readings:

- information items:
 - o *info*
 - *description* (string)
 - *board* (reference to a datapoint describing a HV board)
 - *limits*
 - *vMin* (floating point number)
 - *vMax* (floating point number)
 - *iMax* (floating point number)
 - o *settings*
 - *v0set* (floating point number)
 - *i0set* (floating point number)
 - *switchOn* (boolean)
 - *rampUp* (integer)
 - *rampDown* (integer)
 - *tripTime* (integer)
 - o *readings*
 - *v0* (floating point number)
 - *i0* (floating point number)
 - *isOn* (Boolean)

2.2.1.1. Creating a Datapoint Type

The steps to create the datapoint type with the PARA module are shown in Figure 22.

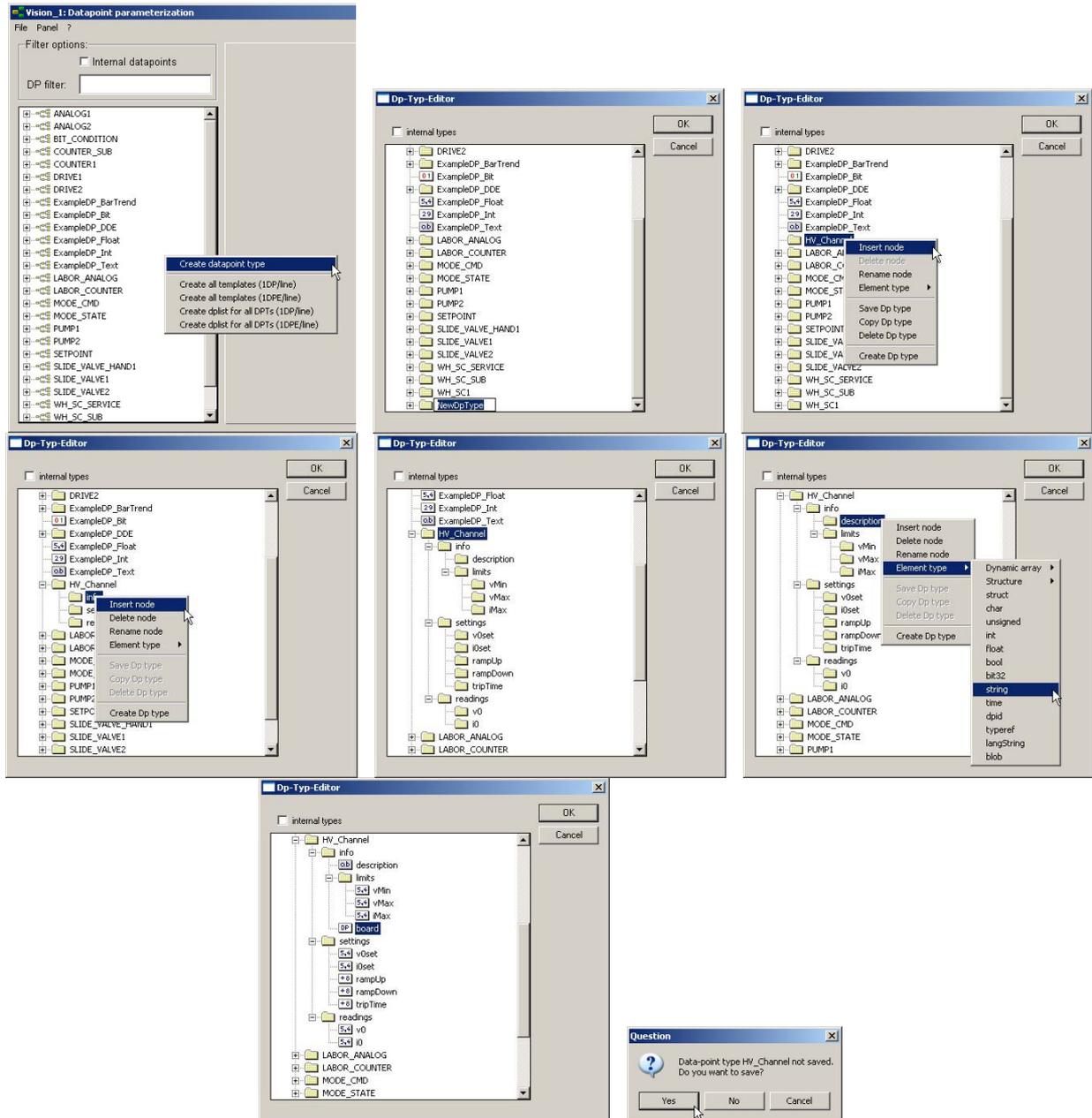


Figure 22 Steps to create a datapoint type

3. First Steps with the JCOP Framework

The following chapter is meant to be a step-by-step instruction for basic framework tasks. It does not replace nor supersede JCOP Framework documentation, furthermore it is by far not complete. Only the basics taught in the course are part of the description.

In order to work with the JCOP Framework in a PVSS project, it needs to be installed in that project (3.1). Afterwards, the usual steps to create a controls system are creating the hardware devices (3.2.1.1, 3.2.1.2, and 3.2.1.4), testing them (3.2.1.3), creating a logical view of the system (3.2.1.5), and correlate devices using finite state machines (3.2.3). The logical view is optional, but facilitates work with the system for non-specialists. Otherwise, the steps can be iterated until the system is complete.

3.1 How to Get and Install the JCOP Framework

The following instructions are currently valid. The always up to date instructions can be found on the JCOP Framework web pages. From there, other components can be found too. For example the ELMB component referred to later is not part of the standard framework distribution. Components like that can be found via the **Developments** link on the download page. The installation instructions are either provided per component (in case of complex installation) or are the same as for the standard ones. If the installation is generic, you can also download the standard set and the specific components and extract them into a common temporary directory as detailed below.

3.1.1. Where to Get the JCOP Framework

Download the JCOP Framework installation tool from the framework pages:

<http://cern.ch/itcobe/Projects/Framework/>

There you will find the JCOP Framework distribution as well as documentation and related links (see Figure 23).

Framework Project

The Framework is an integrated set of guidelines and software tools which is used by Developers of the Control System to develop their part of the Control System application. When all parts of the application have been developed and integrated these form the complete Control System (from a software point of view).

Where do I go from here?	Release Date/ Version	Software	Requires	Release Notes / Highlights
If you do not yet have PVSS				
If you are getting started and need Framework				
If you are looking for documentation				
If you need a Framework course				
If you need a PVSS course				
If you need a Framework hands-on				
If you need a PVSS hands-on				
If you are looking for help				
	27/01/2004	Installation instructions [TXT]		
	2.0.4	Component Installation Tool [ZIP]	PVSS 3.0 RC 1 on Windows or Linux	List of improvements and bug fixes
		Framework 2.0.4 [ZIP]		

Figure 23 Main part of the JCOP Framework download pages.

The first item to download is the ZIP file of the component installation tool. Furthermore you need the distribution ZIP file of the JCOP Framework. Unzip the latter into a temporary directory. You will find up to date instructions in the available Installation Instructions.

3.1.2. A New PVSS II Project Using the JCOP Framework

In order to get started, create a new PVSS project and extract the installation tool ZIP file in top of this directory, using the folder names from the ZIP file and overwriting any already existing files (see Figure 24).

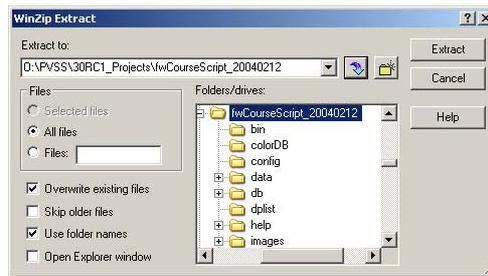


Figure 24 Extracting the JCOP Framework Installation Tool to the PVSS project's folder

Having done that, just start your PVSS project and wait for the graphical editor to start. From there we are going to run the installation tool as shown in Figure 25. It is located in the folder `fwInstallation` and the main panel is called `fwInstallation.pnl`.

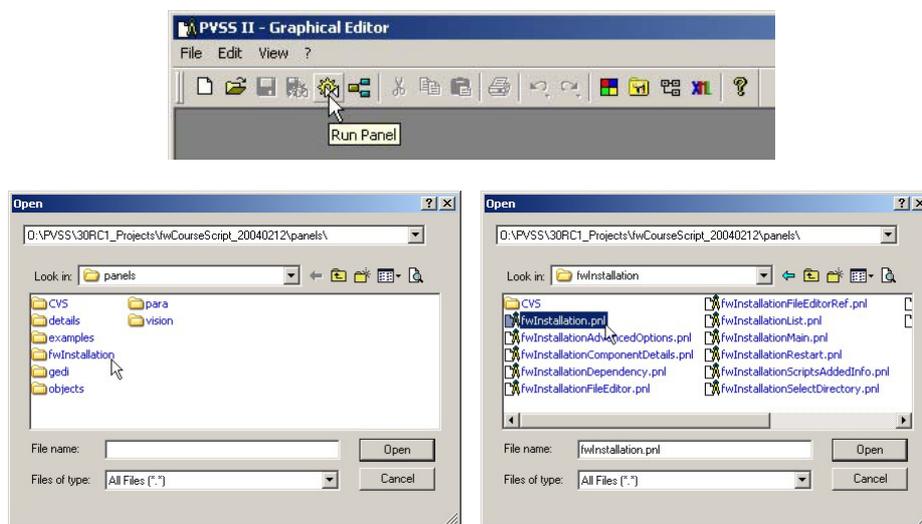


Figure 25 Run the installation tool from the graphical editor.

Starting the tool for the first time in a PVSS project will pop up a dialog box (see Figure 26), where you should enter an installation directory for the JCOP Framework. You can use the same installation directory for all your projects. However, you will always need to install the framework for a new project.

As it is not allowed to have project paths inside project paths in PVSS on Linux, and the JCOP Framework should be installed in a separate project path, it is not advisable even on Windows to choose a sub-folder of your project as installation directory.

Your choice of the installation directory is not final and can always be changed using the Advanced Options button in the installation tool.



Figure 26 Setting the installation directory for the JCOP Framework.

The main panel of the installation tool shows you the components that can be installed on the left, the already installed

components on the right hand side. First you need to choose a source directory, where the tool will search for installable components. In our case, this is the temporary directory where you extracted the JCOP Framework distribution (cf. 3.1.1). This will show you a picture similar to Figure 27.

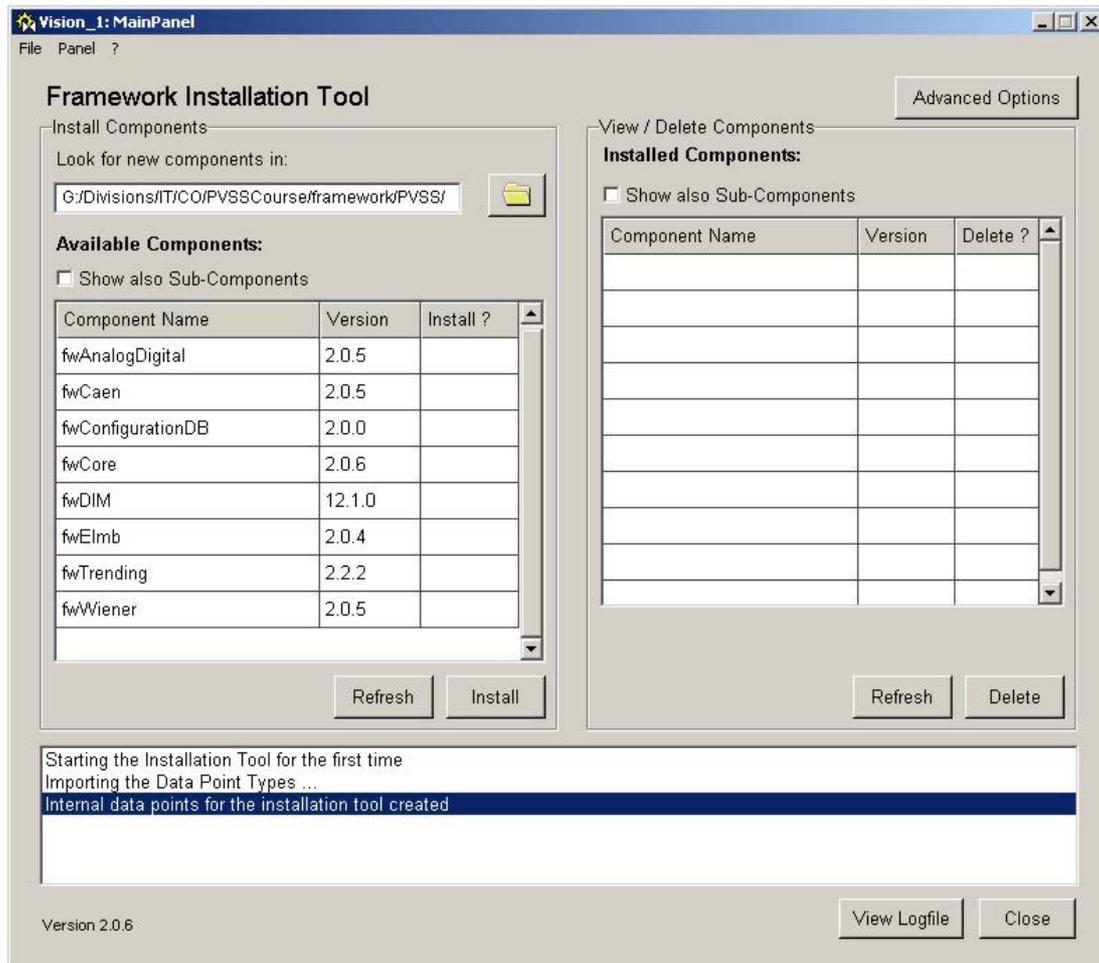


Figure 27 An example for available components for the JCOP Framework.

In the table on the left-hand side you can get additional information about the listed components by double-clicking on their names, or select them for installation by single-clicking in the “Install ?” column. For the further use in the context of the course, please choose the following components for installation and click on Install:

- fwAnalogDigital,
- fwCaen,
- fwCore,
- fwDIM,
- fwElmb¹, and
- fwTrending.

The tool will show you a short overview with your chosen components and the currently set installation directory (see Figure 28). After your choice to install those components, the tool will check dependencies, i.e. if you have chosen components that need others to be installed.

The installation is automatic and takes some time. If components can be installed with several options and/or further steps are needed, the installation tool will pop up a dialog box. In case of the chosen components, it will only remind you that for the CAEN component, you need to add an OPC client driver with number 6 and install the CAEN OPC

¹ The fwElmb component does not come with the standard framework distribution. Please download it from <http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/DCS/ELMB/DIST/ELMBdoc.html> and extract it into the same temporary directory as the rest of the framework distribution.

server. Take note of this and close the dialog box.

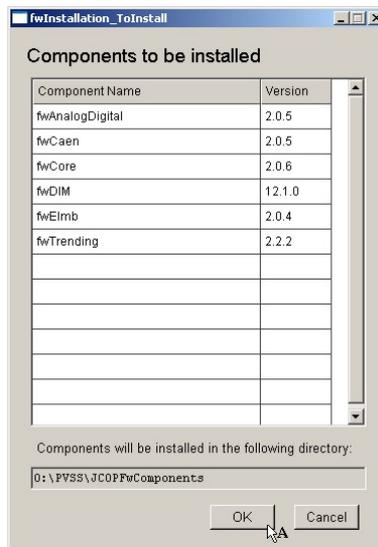


Figure 28 Overview of components chosen for installation.

At the end of the installation, the tool suggests that you add a control manager running the JCOF Framework scripts. Please refer to Figure 29 for the needed parameters.

After the complete installation, you need to restart your complete PVSS project, as the tool has changed your project's config file and this is only read at start up by all managers.

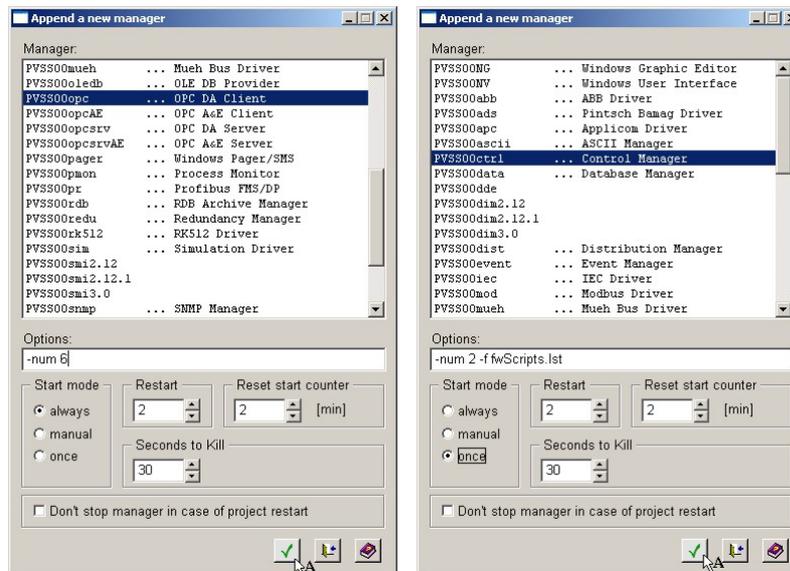


Figure 29 Adding an OPC client driver with number 6 and a control manager for the framework scripts with number 2. Please note that number 6 is obligatory for CAEN devices in the JCOF Framework in order to use the automatic setup, whereas number 2 for the control manager is a choice up to you.

3.2 Starting the JCOF Framework for the First Time

The JCOF Framework tools and devices we are using as developers are mainly concentrated in the Device Editor and Navigator (DEN). This will be the central tool for all developments. It is advisable to add a user interface manager for it with all restart options, so that in case of changes, a quick restart is possible. Append a manager similarly to Figure 29

with the following options:

```
PVSS00NV -p fwDeviceEditorNavigator/fwDeviceEditorNavigator.pnl
```

3.2.1. The Device Editor and Navigator Basics

After that, your DEN will show up with an empty tree of hardware devices (see Figure 30). This tool has two modes, “Editor” which provides configuration capabilities, and “Navigator” which provides basic operation panels for the used devices. It starts up in the navigator mode and can be switched using the buttons below the tree widget. There exist three main views of your controls system, “Hardware” where all devices are listed by so-called “vendor node”², “Logical” where you can build your own view, e.g. by sub-detector, and “FSM” where all finite state machines defined in your experiment are shown.

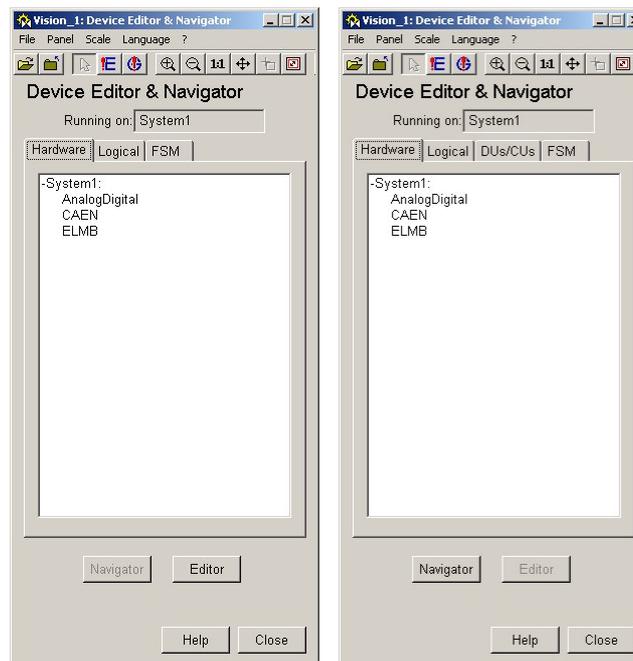


Figure 30 The JCOP Framework’s DEN at first start in a new project, in navigator mode (left) and editor mode (right).

As you can see already here, the JCOP Framework is capable of running on distributed systems. In the current example we are working on System1. Switching to the editor mode will expand the FSM tab by adding a tab to define device unit types and logical unit types, and by this give you the full configuration capabilities there.

Because of some small limitation in the tree widget, you always need to first select a line by left-clicking and then can open a context-menu with a right-click. For example, selecting the system itself will provide a context-menu from which you can start the installation tool again.

Our basic course system will be built “bottom to top” first, in order to create the devices, and “top to bottom” afterwards, in order to integrate them into a complete controls system. As we start “bottom to top”, we will start in the hardware view.

3.2.1.1. Creating Simple Devices – Analog or Digital Inputs or Outputs

Our first step towards a controls system will be the creation of a simple analog input device. Those are generic devices that can be connected to hardware via any of the existing PVSS drivers and represent a single data item, here an analog (floating point) variable. The way of creating this device is generic for most of the device creations from the DEN.

The first step is always to choose the type of device you want to create, here an Analog/Digital device. This will open a device management panel from which you can now create the specific device types (see Figure 31). You can Add or

² “Vendor” may also mean type of devices. Currently, there are “Analog/Digital”, “CAEN”, “ELMB”, and “Wiener”.

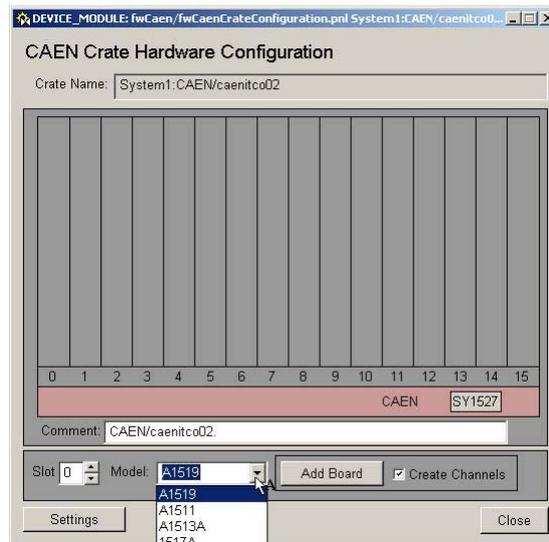


Figure 35 Inserting a board and its channels into a CAEN SY1527 device

Note that the interruption of a creation process for any device will leave the device in most cases not operable, as only part of the datapoint elements are set up. Always wait for the progress bar to end.

3.2.1.3. Operating Devices in the DEN

In order to operate devices from the DEN, one changes into the navigator mode.

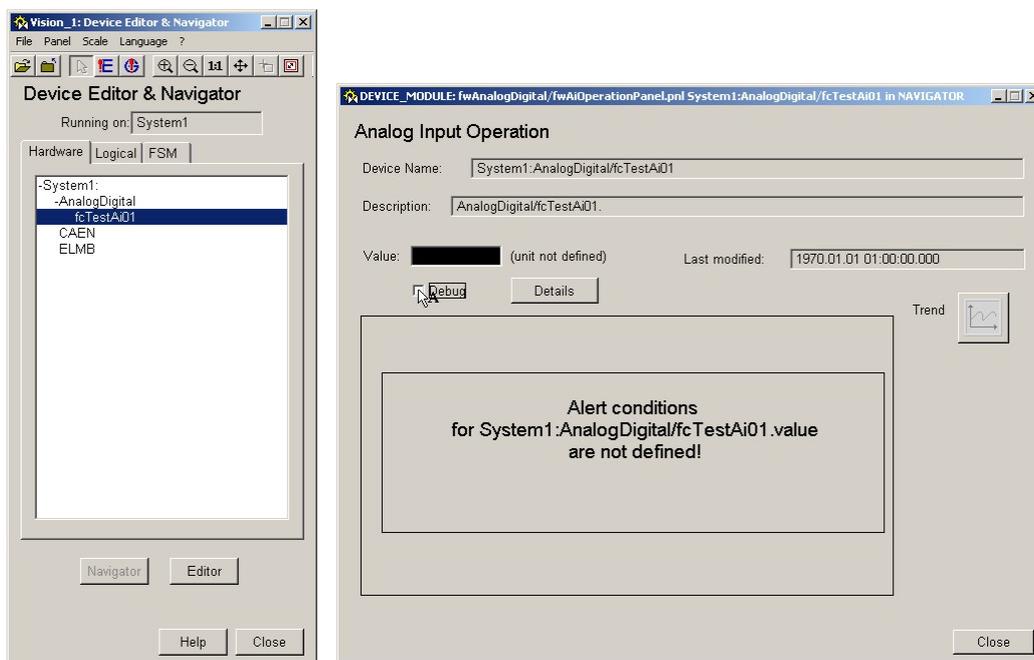


Figure 36 Operation panel for an analog input device

Using a device without hardware connected will set the status of the device to be invalid, which is shown by the black colour in the value field. The Debug mode can be used to set a value to the datapoint element and will set the quality bits accordingly (see Figure 37). In many cases, changes can be done by clicking on the value.

For CAEN devices, special operation panels exist (see Figure 38). These are either synoptics of the physical devices or tables of the most common values. Advanced users can also define own operation panels for their devices.

Operations on the crate will propagate if necessary to the boards and their channels and so on. Standard operations for generic devices are hidden behind the Action button, high voltage operations behind the HV Action button.

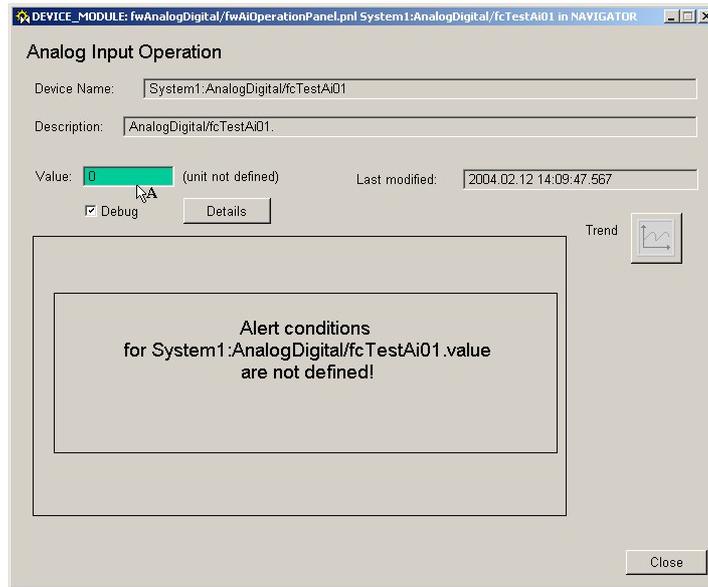


Figure 37 Debug mode for an analog input device

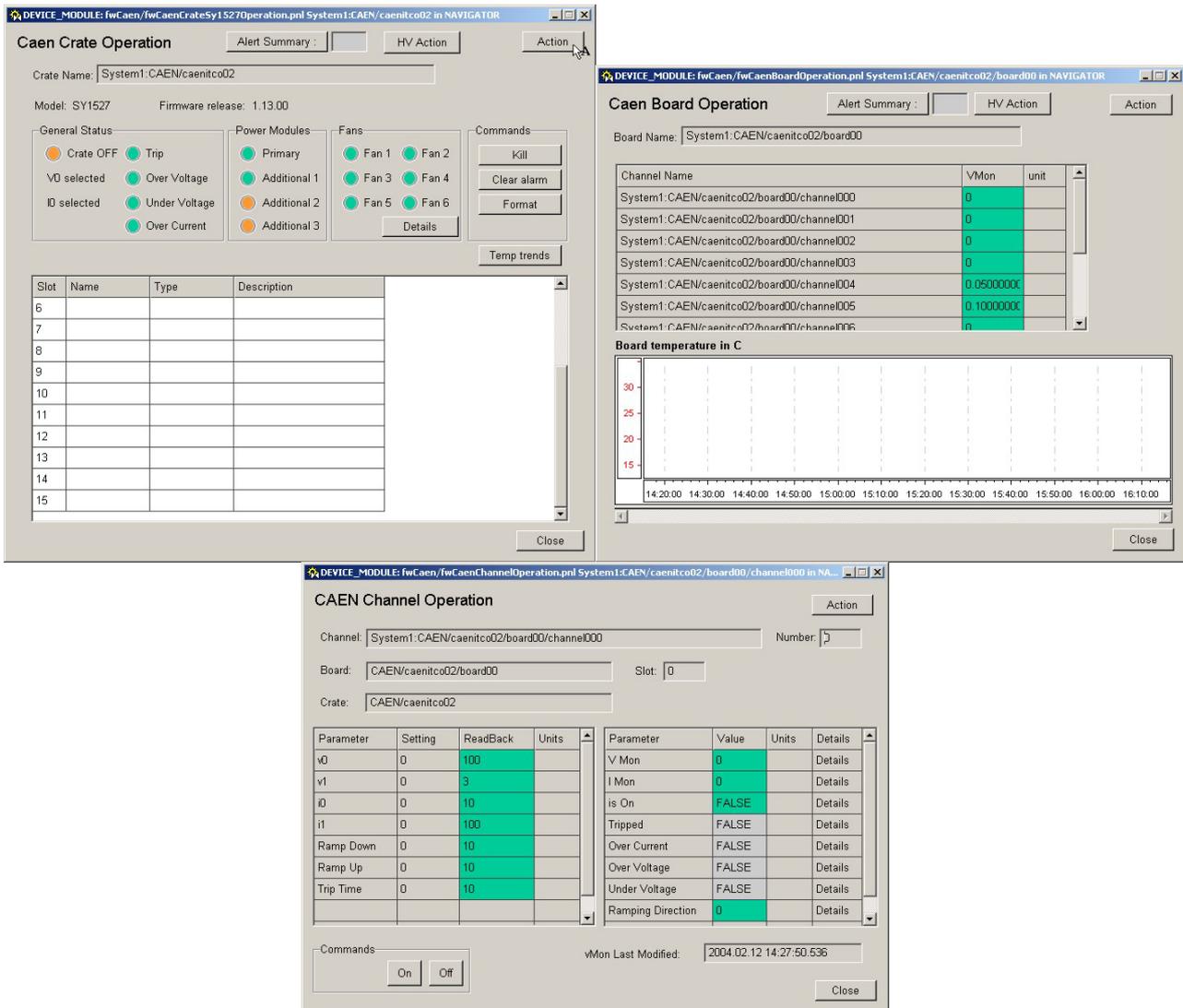


Figure 38 Operation panels for a CAEN crate, board, and channel

In the device action panel, one can mask, unmask, and acknowledge alarms for the device itself and/or its children (see Figure 39).

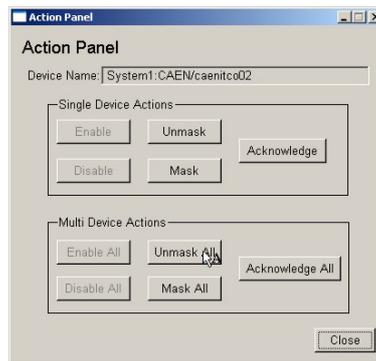


Figure 39 Device Action Panel

For voltage devices, the High Voltage Actions panel exists, from where one can set standard voltage channel parameters for a complete crate or board (see Figure 40).

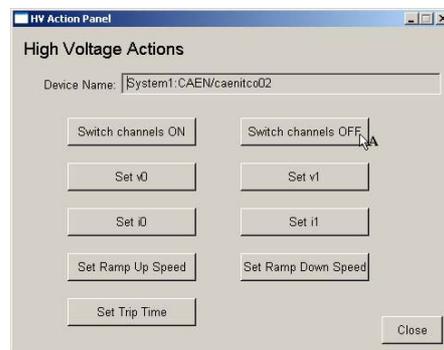


Figure 40 High Voltage Actions panel

3.2.1.4. Expert Configuration of Devices

All devices created in the hardware view have expert configuration panels attached, usually available via the Advanced options button (e.g. Figure 31). From there, options for the datapoint elements can be set, the PVSS configs (see example in Figure 41).

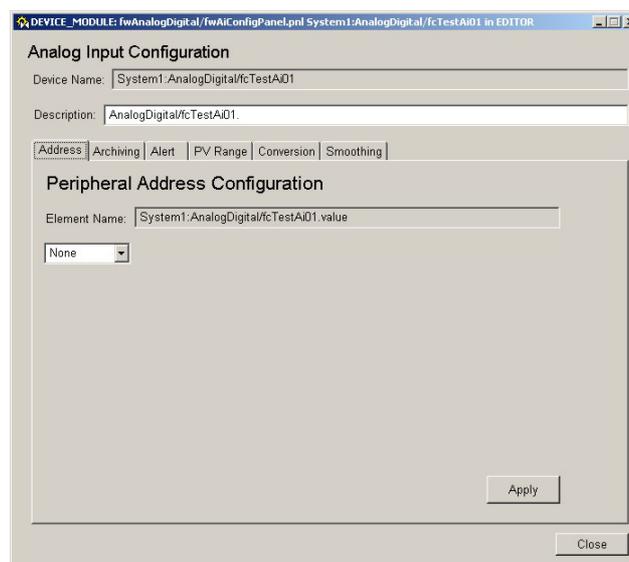


Figure 41 Expert configuration of a framework device

3.2.1.5. Another View of the System – The Logical View

The pure hardware view might be sufficient for a controls system developer with a complete overview of all connections, but it seems necessary to have a second view, following logical lines, e.g. sub-detector and sub-systems. This type of view is suitable for experts of the different domains. An example for the logical view is shown in Figure 42. Without having to know the naming inside the system, the vertex cooling expert could now find the mixed water temperature sensor A, and the high voltage expert the channel connected to wires 1-10 of a chamber.

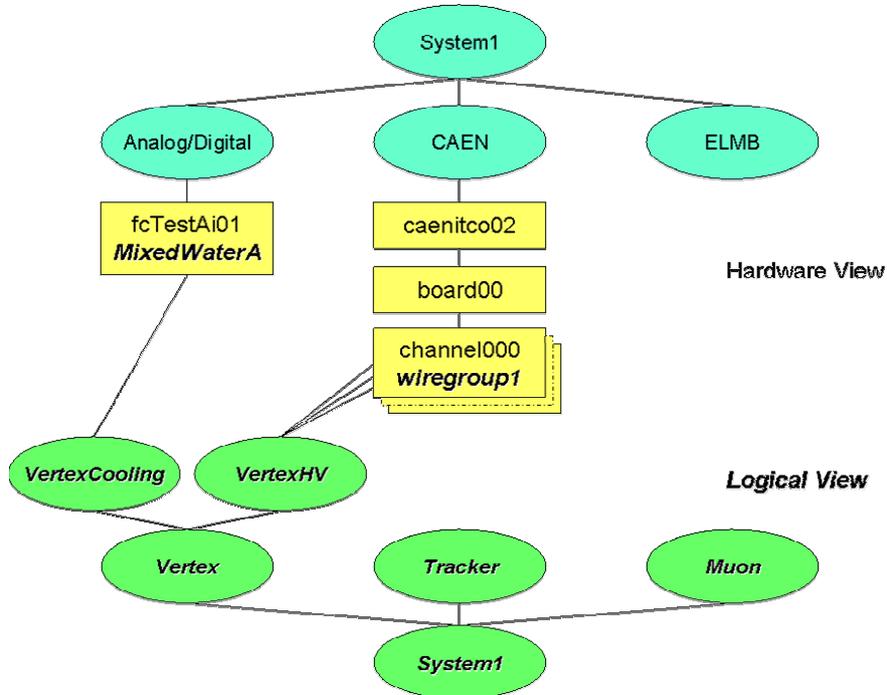


Figure 42 Hardware and logical view example

The steps to setup a logical view are shown in Figure 43 (creation of nodes) and Figure 44 (adding many hardware devices and naming them for the logical view).

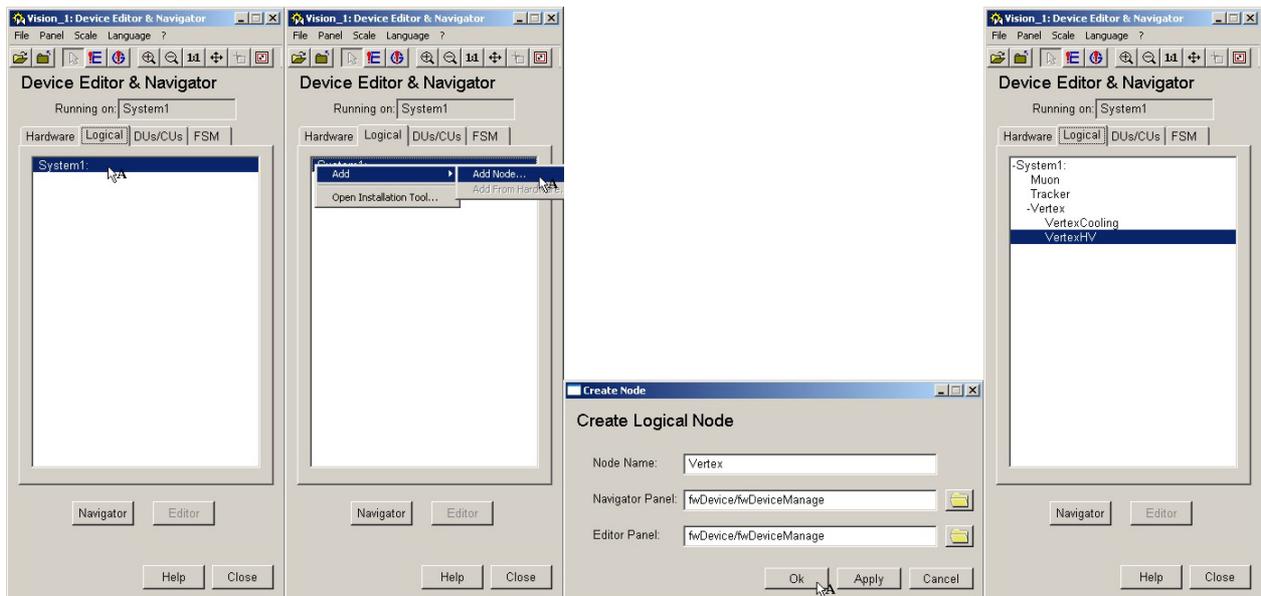


Figure 43 Creation of nodes in the logical view

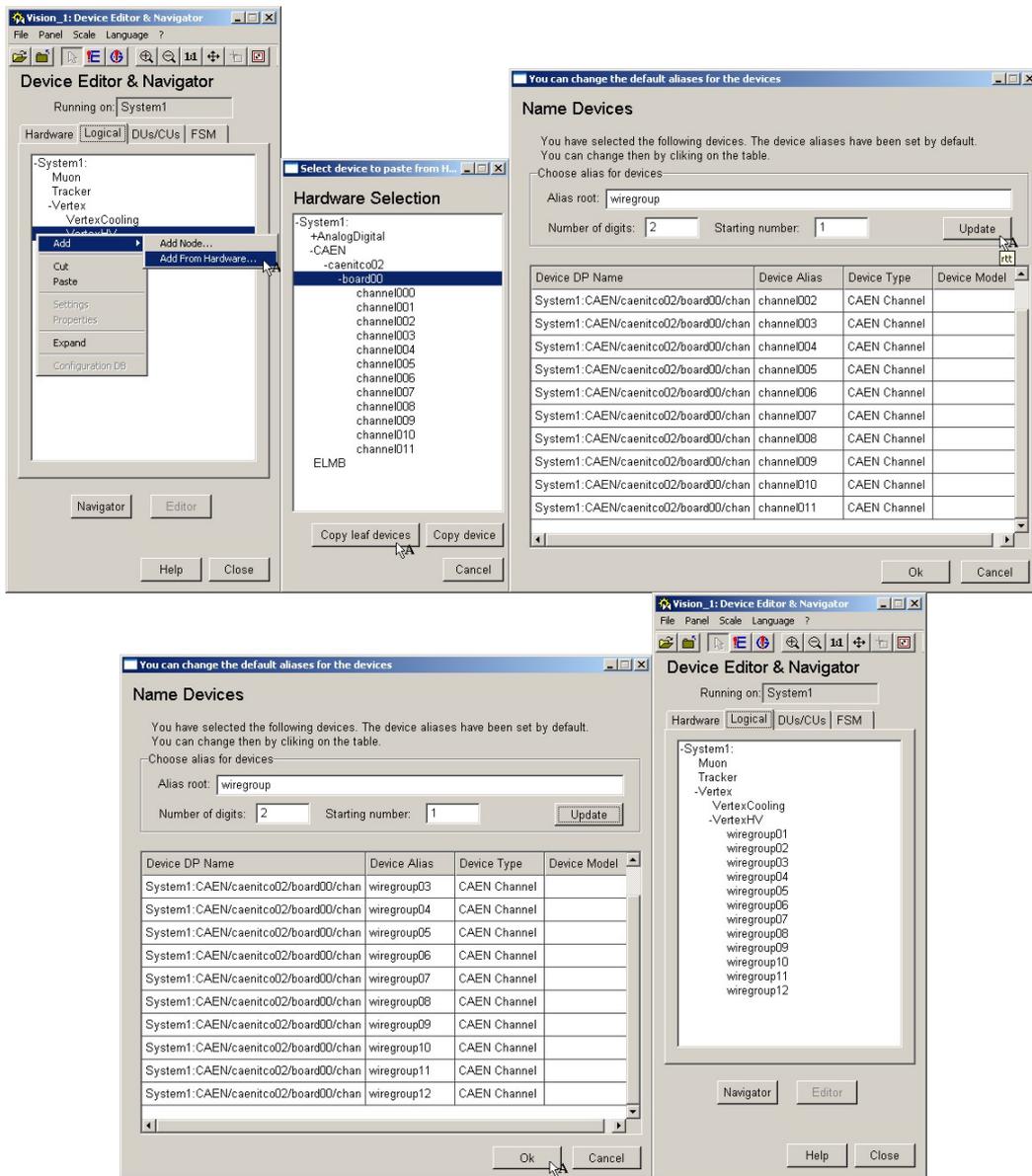


Figure 44 Adding hardware devices to the logical view and renaming them

3.2.2. The ELMB Framework Component

Coming soon

3.2.3. Using FSM with the Device Editor and Navigator

The finite state machine tool integrates the State Machine Interface (SMI++) into the JCOP Framework. Here, only basic operations are explained, for further information please refer to the respective user's guide [4].

In order to use the FSM tool, one needs to add another manager to the project. The steps are similar to those described in Figure 29. The manager needed is a control manager (PVSS00ctrl) and the parameters are:

-num 5³ fwFsmSrvr

Having done this, all functionality of the two tabs in the DEN is available, and Control Units (CU) and Device Units (DU) can be created and organised.

To run the state machines later, one needs to have access to a DIM name server. The communication between the state machines is implemented using the DIM protocol (see [5]). This also implies, that all state machines, that need to connect to each other, have to be know to the same DIM name server.

3.2.3.1. Creating a Control Unit

A control unit is a logical object in the framework user interface. One can create a logical object by clicking on the corresponding button below the list of those objects (see Figure 45). Our first object will be called *DCSNode* and be the generic node type for detector control units. The behaviour we will give to it should be good for most standard sub-detectors and their sub-systems, but does not allow special operations.

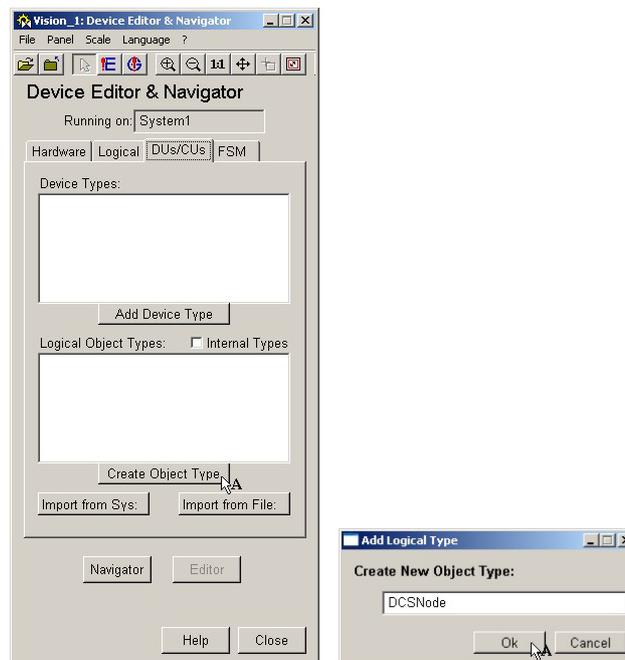


Figure 45 Creating a logical object.

After the creation, the logical object type is shown in the list and the configuration panel (Figure 46 left) can be opened by double-clicking. From there one can open the simple configuration panel (Figure 46 right) via the corresponding button.

Note that the JCOP Framework guidelines [2] contain definitions for state colours and in general also for state names.

³ The manager number can be chosen according to your system and is not fixed.

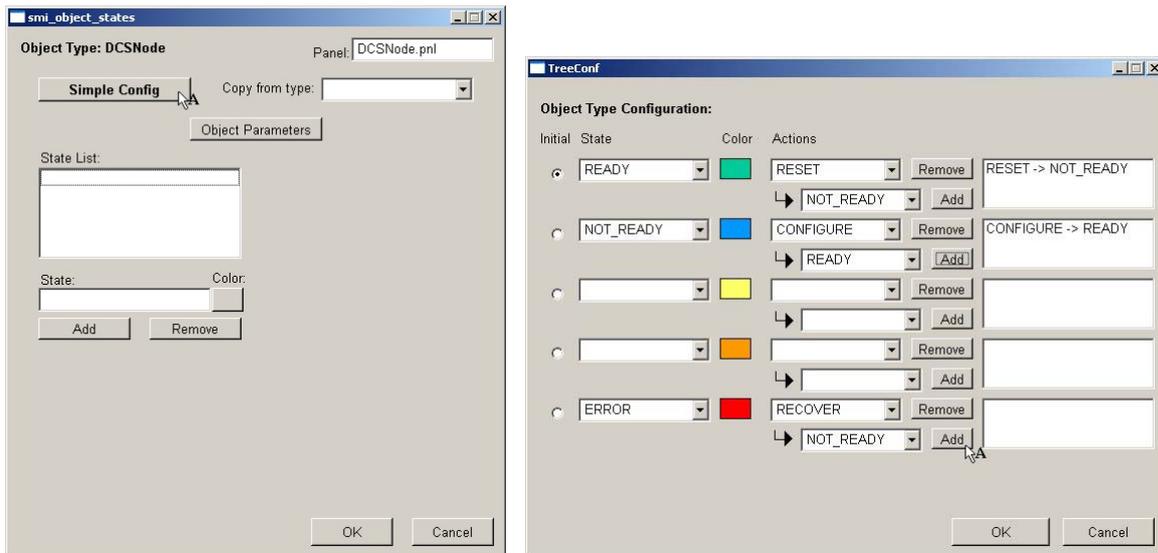


Figure 46 Configuration of a logical object type (standard left, simple configuration utility right).

In the simple configuration, one can give up to five states related to the different levels of severity, either by choosing from the drop-down menus, or by typing in a new meaningful short name for the state. An initial state can be chosen with the radio buttons on the left hand side. In addition, actions to be available from the defined states can be added. The states and actions for our *DCSNode* are detailed in Table 1.

If additional states or actions are needed, they can be set in the standard configuration panel. An example for this can be found in 3.2.3.2.2.

Note that you always need to define complete state machines, i.e. all possible and/or wanted states need to be reachable via the defined actions, and all possible state changes need to be defined in the so-called WHEN clauses (see below). These WHEN clauses are evaluated at start-up of the state machine and whenever the state of a child changes.

Table 1 States, actions, and WHEN clauses for the example *DCSNode*.
 Note that the order of the WHEN clauses is significant!

State	Action	WHEN Clause
READY	RESET ⇒ NOT_READY	1. any child in state ERROR ⇒ ERROR 2. any child in state NOT_READY ⇒ NOT_READY
NOT_READY	CONFIGURE ⇒ READY	1. any child in state ERROR ⇒ ERROR 2. all children in state READY ⇒ READY
ERROR	RECOVER ⇒ NOT_READY	1. any child in state ERROR ⇒ ERROR

These are just the proposed standard states, actions, and WHEN clauses for a new logical unit. For learning purposes they are sufficient, even though obvious state like “RUNNING”, “PHYSICS”, etc. are not included.

3.2.3.2. Giving Behaviour to a Device – A Device Unit

Devices in the JCOP Framework are a priori input/output channels to real world devices. In order to use them inside a hierarchy, they need to have states, alerts, and sometimes actions defined. All framework device type as well as any other PVSS datapoint type can be the basis of a device unit type, the behavioural model for a framework device.

3.2.3.2.1. A Simple Device – A Water Temperature

In this context, a simple device only has states and alerts. The alerts are defined via the hardware view and belong to the instance of the device type. The states on the other hand can be defined for classes of device types, i.e. a device type can be used as the basis of one or more device unit types. In the example, a water temperature device unit type, called *FwAiMixedWater*, is created based on an analog input framework device type (*FwAi*). One can imagine several different device unit types based for water temperatures, e.g. cold water temperature, ... , that only differ by the conditions for their states.

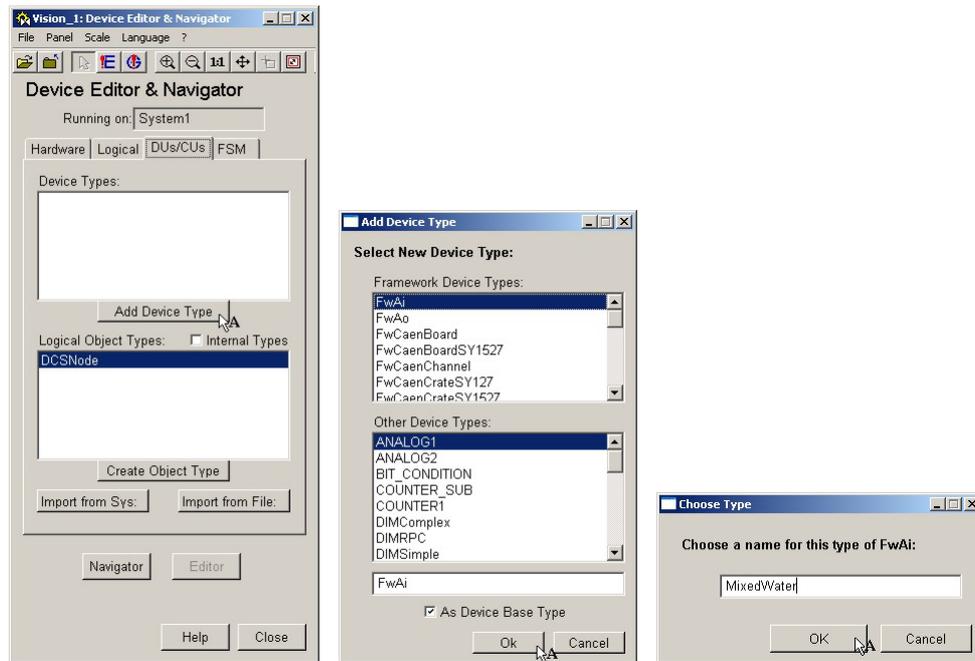


Figure 47 Steps to create a device unit type. Note that one should check “As Device Base Type”.

Clicking the Add Device Type button opens a selection panel for the existing device types in the system (see Figure 47). The name that is given to this device unit type, will consist of the entered name with the name of the base device as a prefix. In the example it will be *FwAiMixedWater* and should appear in the Device Types list. Double-clicking a device type in that list will open the device unit type configuration panel (see Figure 48), similar to the logical object configuration panel shown in Figure 46. Again, one can open the simple configuration utility from there.

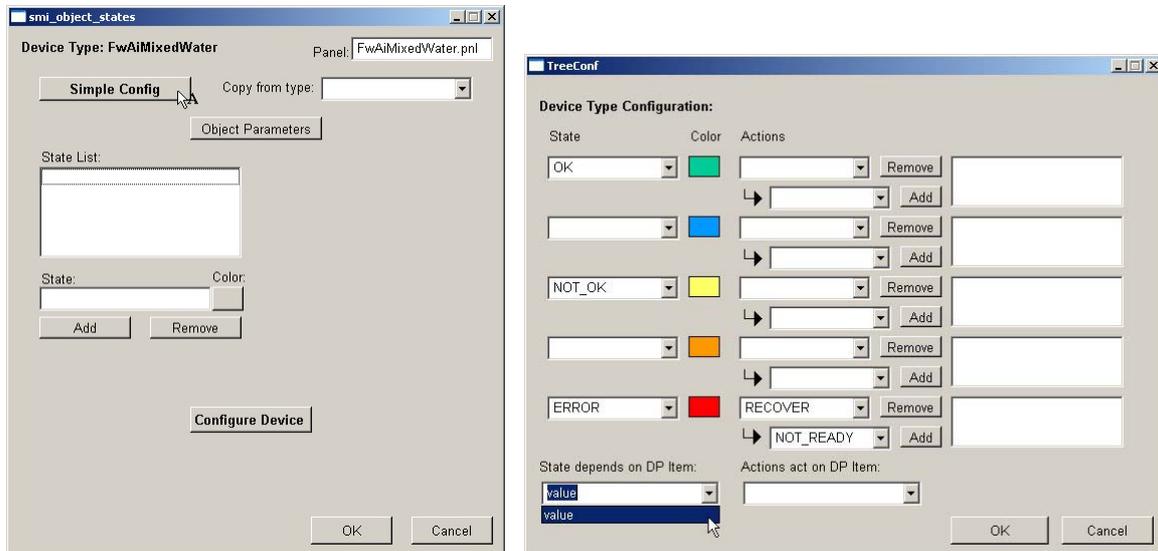


Figure 48 Configuration of a device unit type (standard left, simple configuration utility right).

Apart from the states and actions, one can configure the datapoint element of the device type on which the state depends in the first place and the one, the actions act on. These are first choices, that can be changed later in the course of the specific configuration. Note also that there is no choice for an initial state, as the state solely depend on the device itself. In the example case, no actions are foreseen, the states and their conditions are shown in Table 2.

On the configuration panel of the device unit type, there is no field for WHEN clauses, but a button (see Figure 49) from which one can choose which behaviour should be modified. In fact, there are three PVSS scripts, that define the behaviour, for:

- initialization (*FwAiMixedWater_initialize*),

- state definition (*FwAiMixedWater_valueChanged*), and
- action definition (*FwAiMixedWater_doCommand*).

Those scripts can be edited either with a simple configuration option or directly from that button (see). The parameters passed can be found in the generated script text, but always start with the `domain` (name of the parent) and the `device` (name of the device itself).



Figure 49 Configuration possibilities for device unit types.

In the example, the simple configuration (see Figure 50) is sufficient, the following example (3.2.3.2.2) will show how to program the scripts directly.

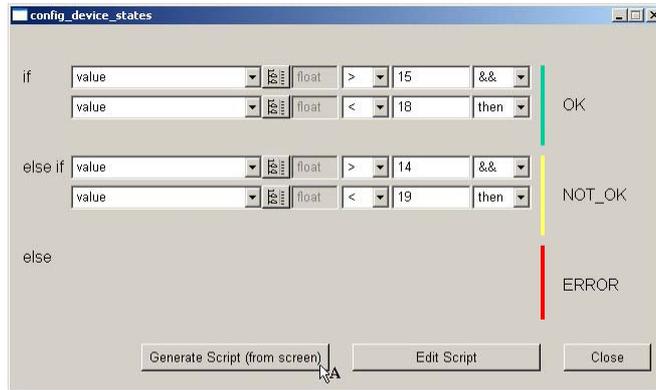


Figure 50 Simple state configuration panel

The **Generate Script** button will generate a PVSS script from the screen, which could be checked via **Edit Script**. Note that generating a script from the panel will overwrite any existing script for this device unit type.

Table 2 States and Conditions for the MixedWater Device Unit Type

State	Severity Level ⁴	Condition
OK	OK, Physics Datataking	15 < value < 18
NOT_OK	First level of error severity	14 < value <= 15 18 <= value < 19
ERROR	Third level of error severity	else

As the device unit type has no actions, it is now configured. Refer to the complex example in order to have a more complete device with states and actions.

3.2.3.2.2. A Complex Device – A Voltage Channel

The basic steps to create a more complex device are more or less the same, so Figure 51 only shows a few screenshots. Afterwards, in order to add actions and parameterize the states, additional steps are necessary. There is an additional field to define on which datapoint element the actions will manipulate the device.

In case of a JCOP Framework CAEN channel representation, the state can be based on the datapoint element `actual.status`, whereas the action can toggle `settings.onOff`. Starting from this status word, flags can be evaluated.

Table 3 States and Commands for a HV Channel

State	Severity Level ⁴	Condition	Actions
ON	OK, Physics Datataking	bit 0 set (channel on)	SWITCH_OFF
OFF	OK, No Physics Datataking	no bits set	SWITCH_ON
RAMPING	First level of error severity	bit 1 or bit 2 set	SWITCH_OFF

⁴ These levels are defined in [1].

			SWITCH_ON
OVERCURRENT	Second level of error severity	bit 3 set	
TRIPPED	Third level of error severity	bit 8 or bit 9 set	RECOVER
ERROR	Third level of error severity	else	RECOVER

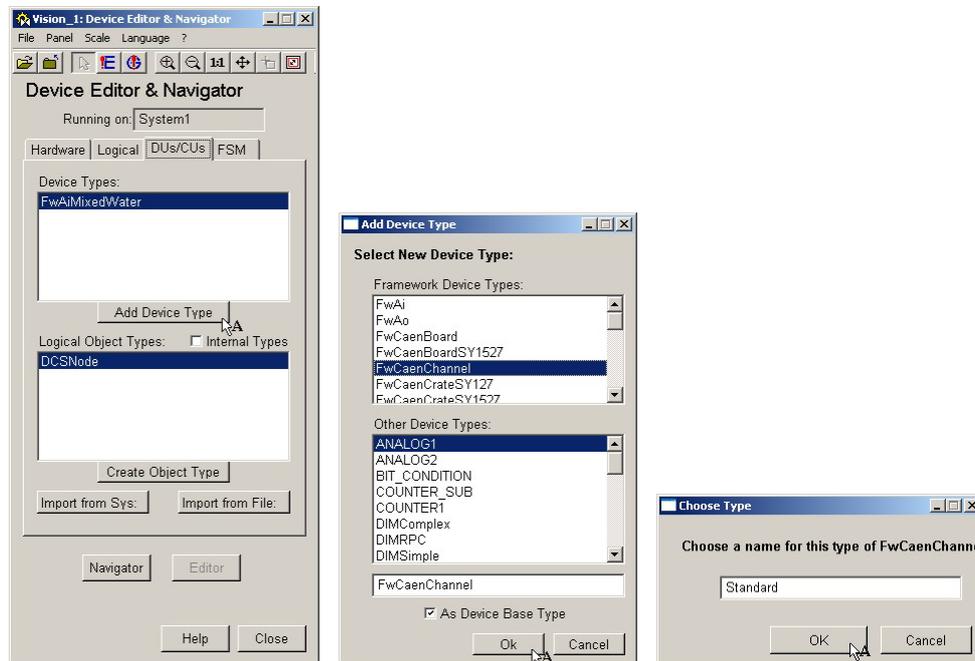


Figure 51 Basic creation steps for the complex device unit type

Having set up the basic states and their corresponding commands (see Figure 52), one can now include more states using the steps shown in Figure 53. This should be repeated also for another state called “*OVERCURRENT*”, which is a warning. All in all this results in the state and command matrix shown in Table 3.

Of course, this is just a simple example, one could imagine far more sophisticated models for a HV channel, obviously here, *RAMPUP* or *RAMPDOWN* as well as other status information of the device could be taken into account.

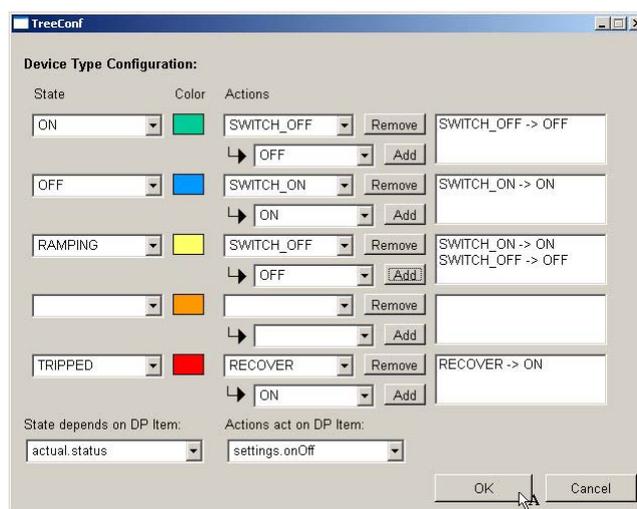


Figure 52 Basic state setup for a voltage channel

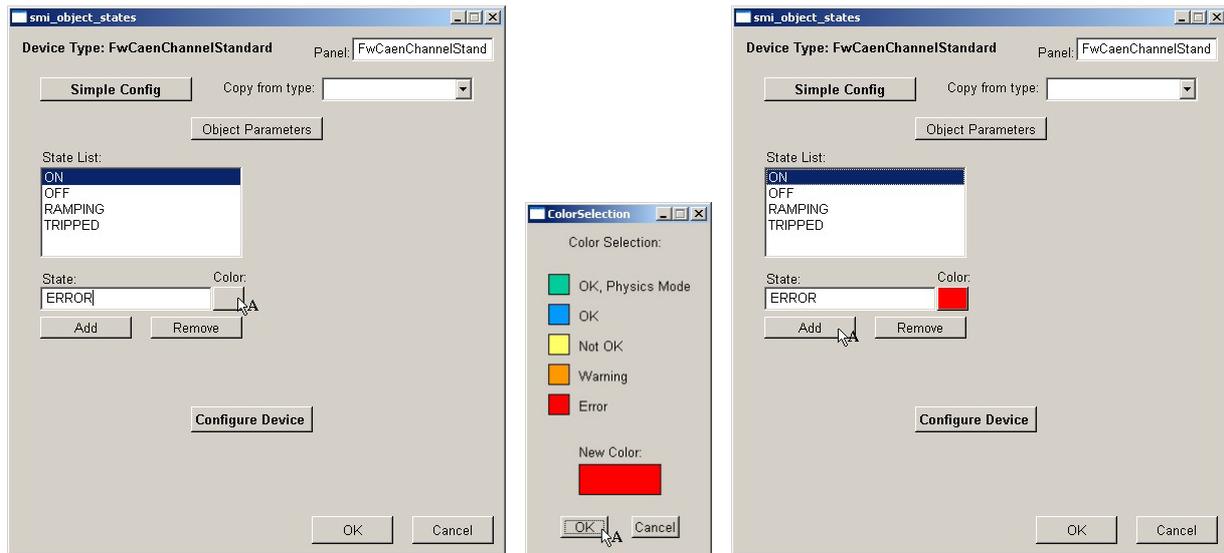


Figure 53 Steps to include a new state

In order to now define the behaviour of the device unit type, the simple configuration is only sufficient for the actions, the states need to be evaluated from a single status word.

Because of the specific model of a CAEN voltage channel, a trip state is cleared by switching the channel back on. So only the settings.onOff datapoint element is sufficient:

- SWITCH_OFF sets the bit to 0, and
- SWITCH_ON and RECOVER set the bit to 1.

Table 4 shows an example script for the state evaluation of a CAEN voltage channel. It implements all states listed in Table 3. Note that also here, the order is crucial. One could implement all other possible states by evaluating the currently not used bits in the status words.

Table 4 Standard script for a device unit type of a CAEN channel.

```

FwCaenChannelStandard_valueChanged( string domain, string device,
    int actual_dot_status , string &fwState)
{
    bool isOn = getBit(actual_dot_status, 0);
    // only ramping is defined here, but the bits 1&2 denote ramping up and down
    bool ramping = (getBit(actual_dot_status, 1) || getBit(actual_dot_status, 2));
    bool OvC = getBit(actual_dot_status, 3);
    // bits 4-7 denote other error states, e.g. over voltage
    bool Trip = (getBit(actual_dot_status, 8) || getBit(actual_dot_status, 9));
    if(Trip)
    {
        fwState = "TRIPPED";
    }
    else if(OvC)
    {
        fwState = "OVERCURRENT";
    }
    else if(ramping)
    {
        fwState = "RAMPING";
    }
    else if(isOn)
    {
        fwState = "ON";
    }
    else if(actual_dot_status != 0) // additional unknown bits are set
    {
        fwState = "ERROR";
    }
    else
    {
        fwState = "OFF";
    }
}
    
```

3.2.3.3. Translations between Parents and Children

At some stage in the hierarchy, a divide exists, usually even more than one. These are the connection points between parents and their children, that do not have the same states and/or actions. If the system is planned coherently before, this does not happen frequently. One way to tackle this problem is to introduce additional layers of logical units that translate between parents and their children. This is not overly performant, so one should aim for a coherent set of interfaces between parents and their children from the beginning, i.e. having special logical units with all the behaviour and the understanding of their children and parents built in.

In the example here, there are two device unit types that need to be connected to control units, the mixed water temperature and the voltage channel.

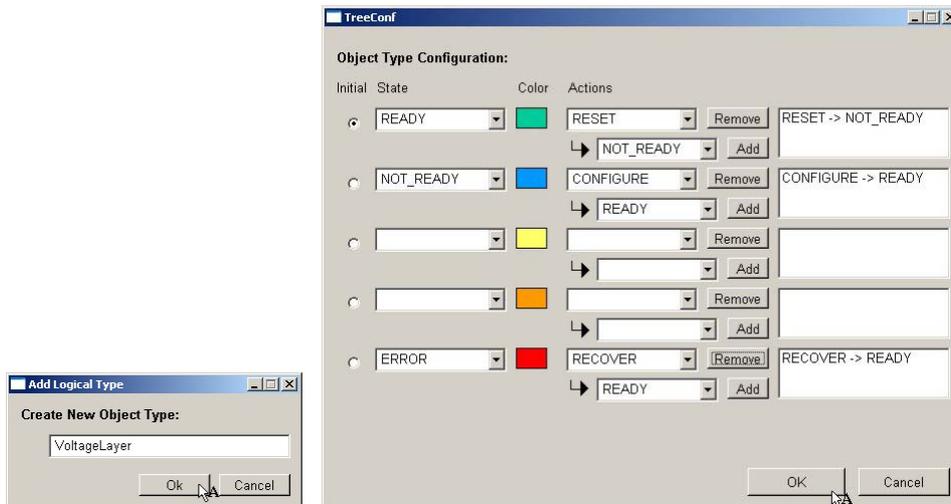
For the water temperatures, a cooling control unit is constructed, that only has states (known to the *DCSNode*) derived from those of the children (*FwAiMixedWater*). No actions are known to this control unit.

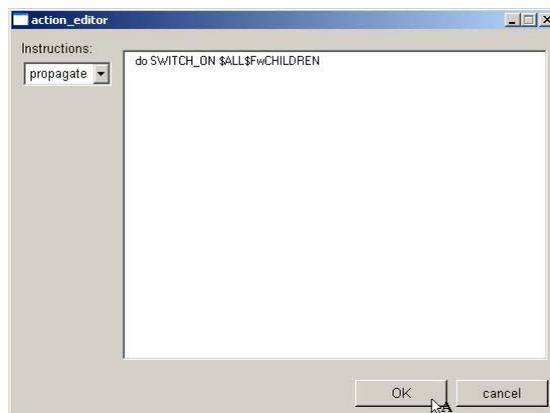
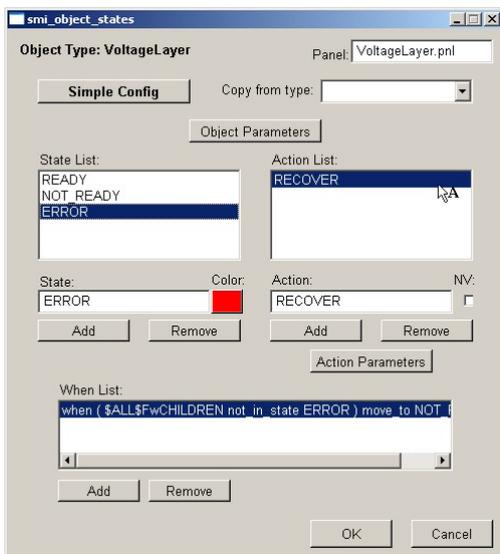
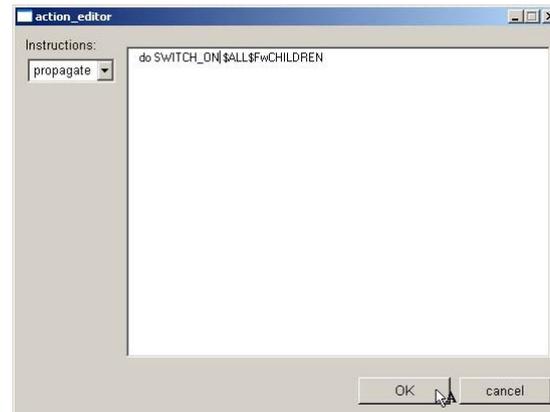
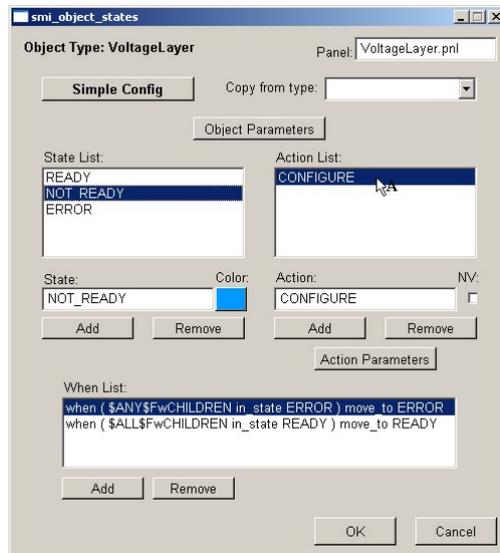
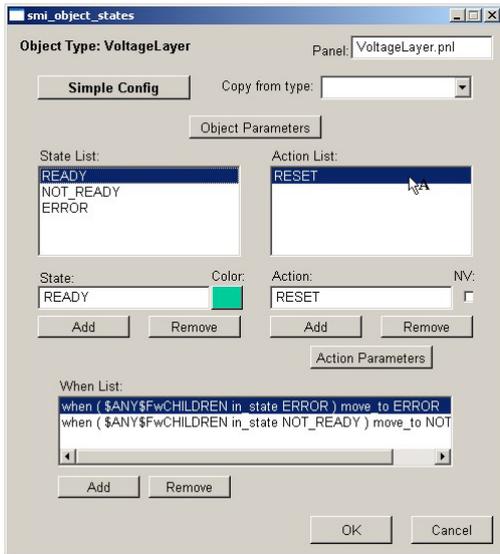
The more complex example, that is shown completely here, is the *VoltageLayer*, that collects the states from its children (*FwCaenChannelStandard*) and derives its state from them, and gets commands from its parents (*DCSNode*) and sends derived commands to its children. The states and actions involved are shown in Table 5.

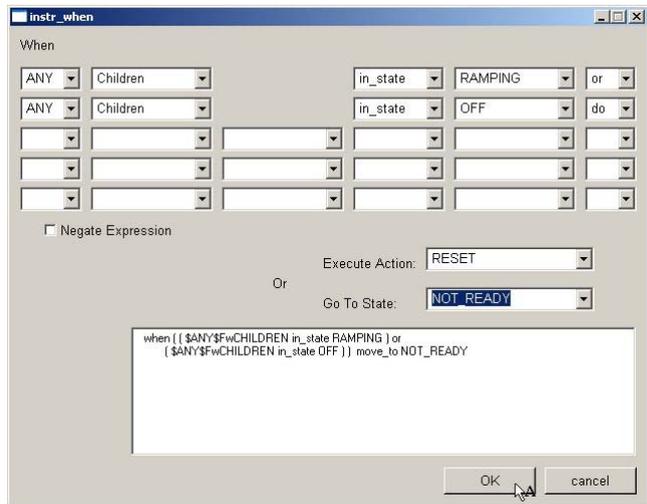
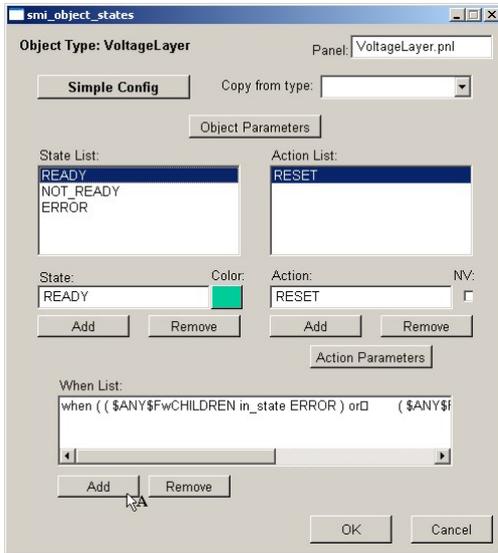
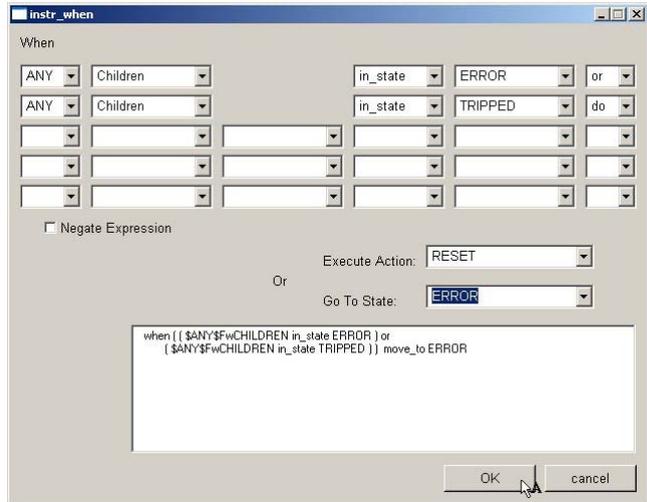
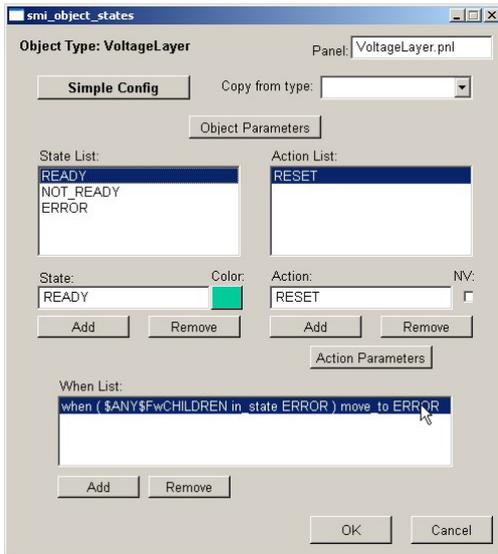
Table 5 States and actions of a VoltageLayer

Parent Command	Command to Children	Child State	VoltageLayer State
CONFIGURE	SWITCH_ON	any in ERROR	ERROR
RESET	SWITCH_OFF	any in TRIPPED	ERROR
RECOVER	RECOVER	all ON	READY
		else	NOT_READY

The following figures show the setup steps for these states and actions.







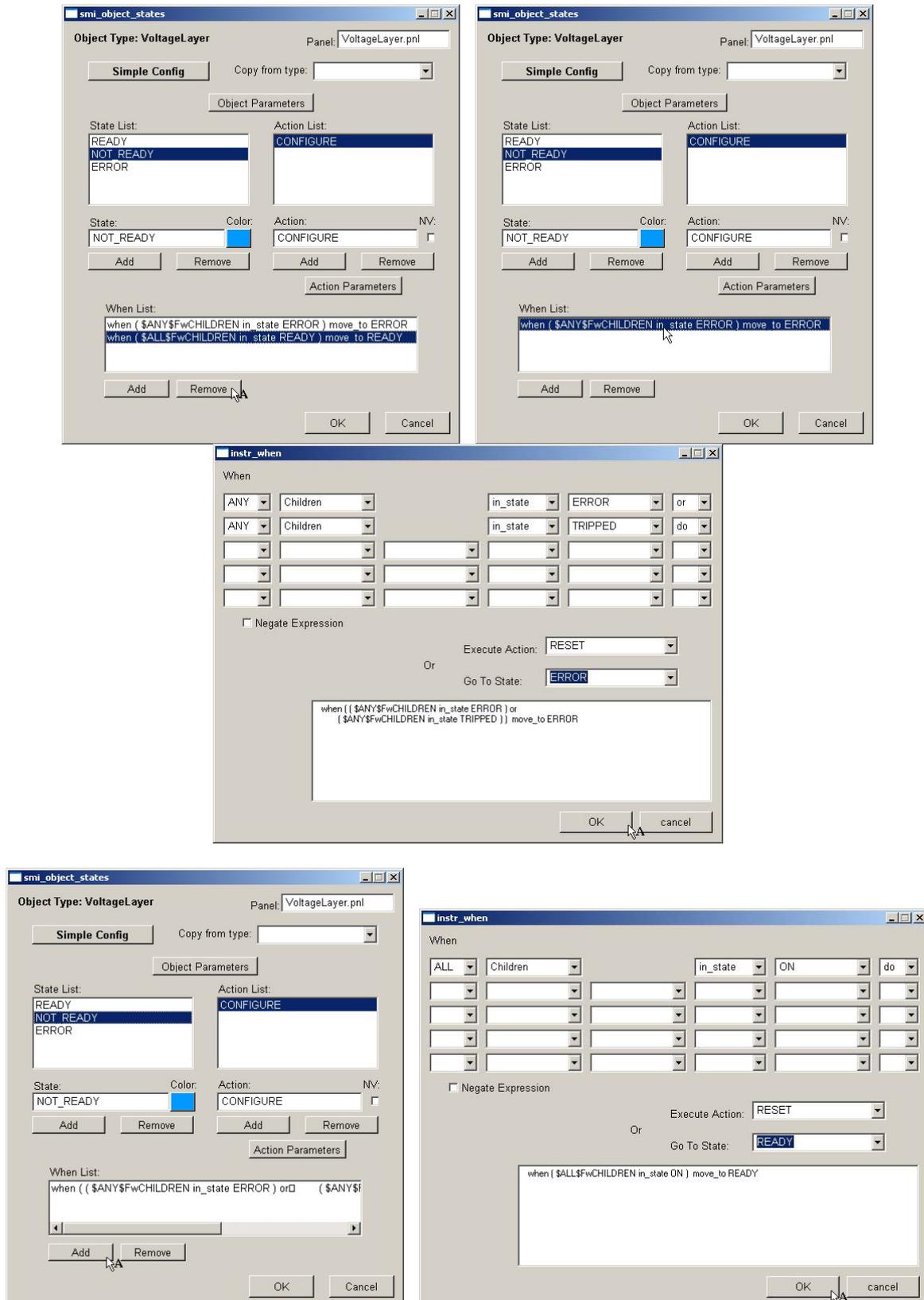


Figure 54 Steps to configure the VoltageLayer

3.2.3.4. Assembling and Operating a Hierarchy

Up to now, the ingredients of a controls hierarchy (control unit types and device unit type in 0, as well as devices in 3.2.1) have been defined. Now, the hierarchy can be assembled by instantiating control and device units. Control units

by definition do not point to something, whereas device units are directly linked to hardware or software devices. Using the JCOP Framework implementation, an unlimited number of hierarchies can be built in parallel. For configuration of these hierarchies, one changes to the FSM tab in the editor mode of the DEN. This implementation makes use of SMI++ [6] that communicates via the DIM [5] protocol, which needs a common DIM name server (DNS). Names of nodes in trees connected to the same hierarchy need to have unique names, as all the addressing only uses the name; furthermore, all control units form a so-called DIM service and like that also need unique names if connected to the same DNS.

3.2.3.4.1. Logical Nodes – Control Units

Any hierarchy starts with a root node. The steps to create one is shown in Figure 56. In this example, a simple hierarchy (see Figure 55) will be built in two steps, first the control unit hierarchy from the top node “myDCS” to the sub-detector level, later the integration of the existing devices. This represents the way a control system is ideally built, defining the structure top-bottom and then integrating devices bottom-top.

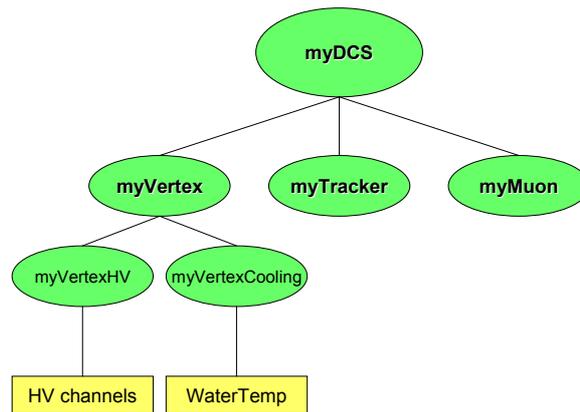


Figure 55 Generic control system architecture (FSM view)

In order to populate the tree from myDCS downwards, the sub-detectors “myVertex”, “myTracker”, and “myMuon” need to be attached (see Figure 57).

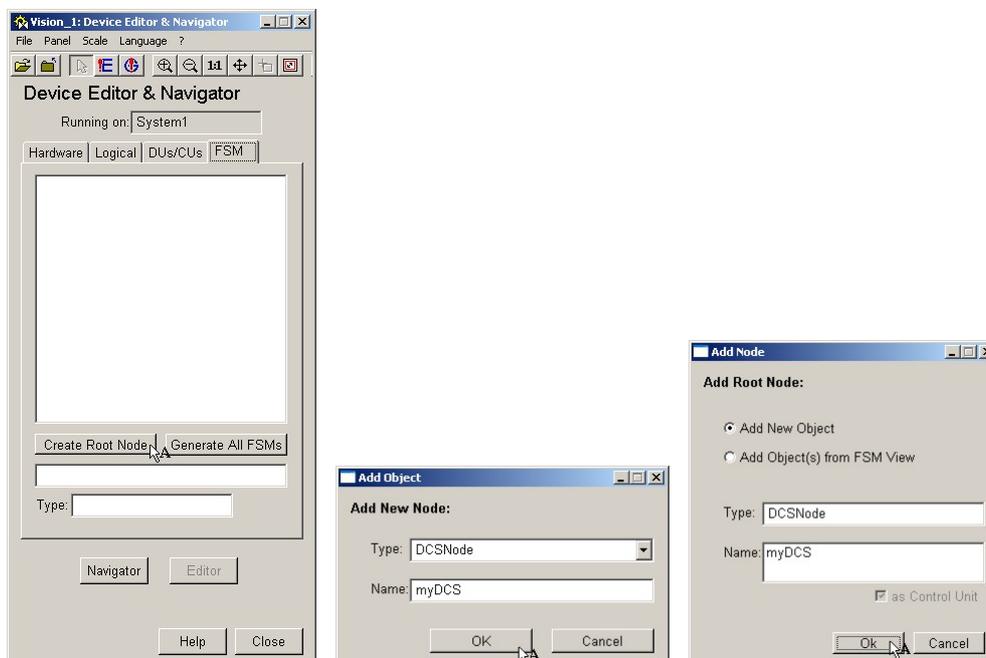


Figure 56 Steps to create a root node

After that, in a first step one can test the hierarchy using only the created control units. Especially as many communication paths are involved, it is recommended to create hierarchies in steps and check often, if the assembled part behaves as wanted. In bigger systems, several hierarchies of sub-detectors or even their sub-systems will be combined to the experiment’s hierarchies.

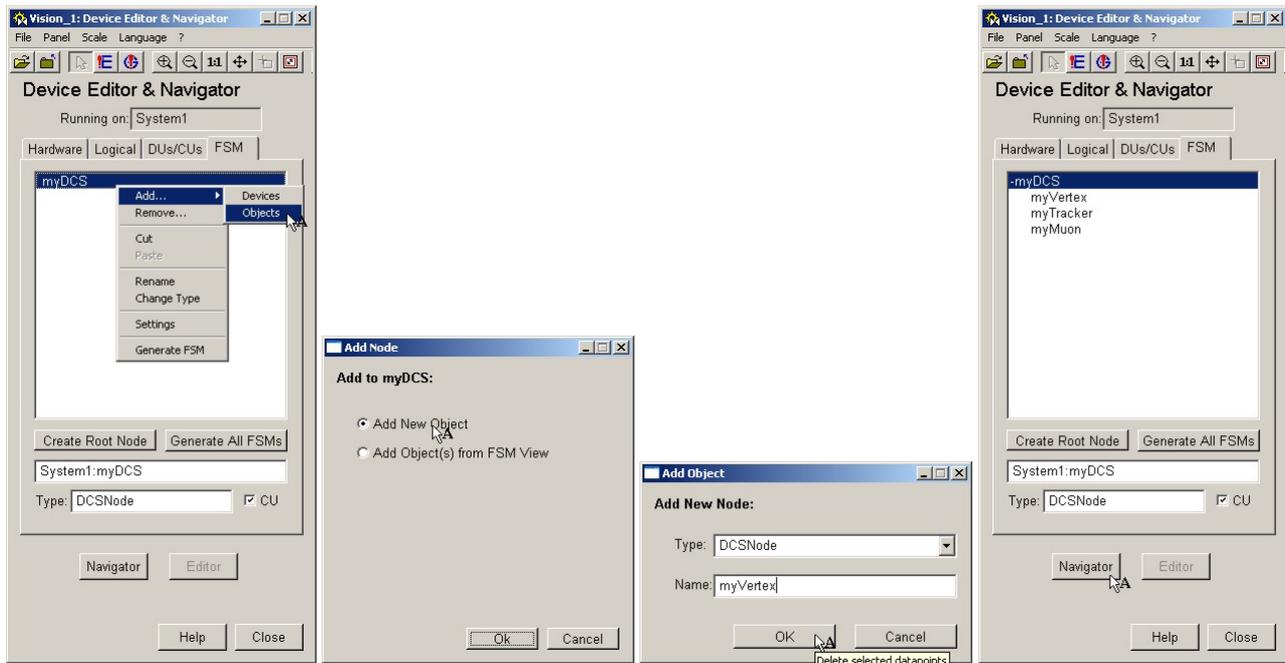


Figure 57 Adding nodes to a hierarchy

In order to run the hierarchy, one changes to the navigator mode and there into the FSM tab. This tab shows all defined hierarchies as well as the chosen DIM name server (see Figure 58). Make sure, a DNS process is running on this node. Otherwise, stop all domains of the hierarchies, either start the DNS task or choose another computer where a DNS runs, and start all domains again.

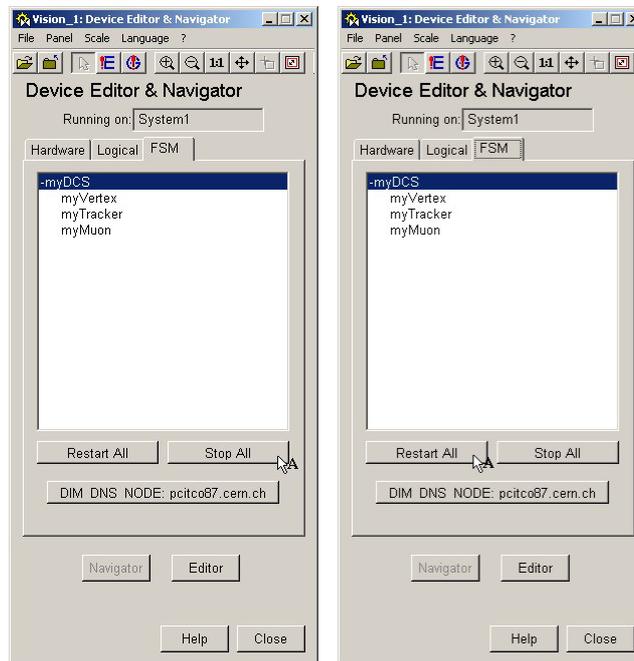


Figure 58 Stopping and starting the hierarchy

After starting the domains, one can view a hierarchy starting from any control unit (domain) in the tree. The context menu on those allows to choose “View” which will open the corresponding operation panel (see Figure 59). Just opening the operation panel will not show state colours for control units but their state names, because the current user is just a spectator, who has no direct access to the control system nodes. The current ownership relation is shown in form of the lock symbol right to the state of the unit. In general, an open lock means that nobody controls this unit currently, whereas a closed lock show that the unit is taken. Further colour coding shows in more detail who the current owner is, relative to the user himself.

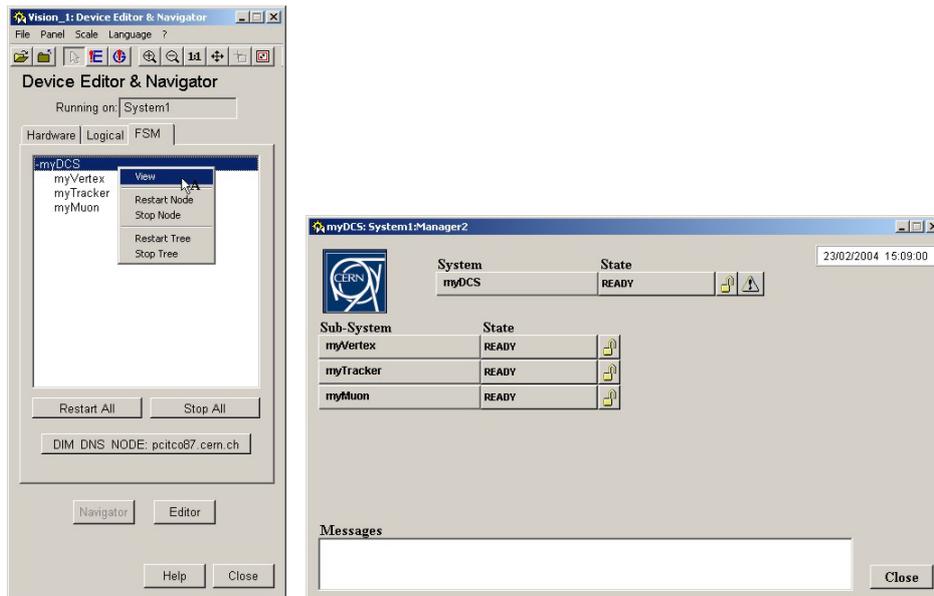


Figure 59 Viewing a hierarchy

In order to be able to operate the system, the user has to take control of a control unit, i.e. attach his user interface as control interface to it. Clicking on the lock symbol will open a small dialog, where the user can **Take** the control of the specific unit. Afterwards, also the state colours are shown, and the user can operate the control unit and its children. Also now the user can **Share** or **Exclude** the device or set it to **Manual** or **Ignore** mode (see Figure 60).

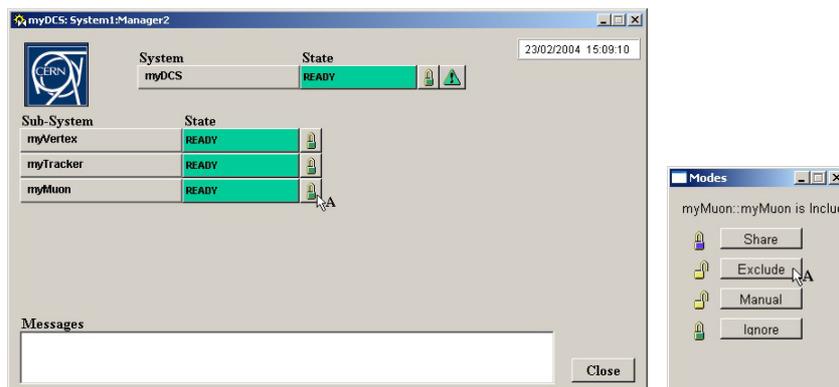


Figure 60 A taken control unit and the corresponding ownership options

All defined options for the control unit type *DCSNode* are now available and can be tested from either the top node *myDCS*, or its children (see Figure 61). The end of the tests should be returning the control of the top node.

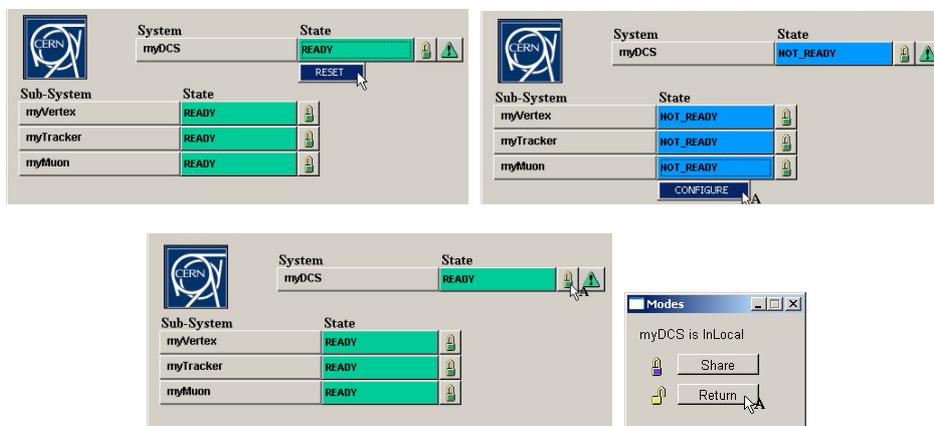


Figure 61 Simple operations with the DCSNode-type control units

Now that this upper part of the hierarchy is verified, one can include the devices and related control units. Starting with the voltage sub-system of the vertex detector, the first steps are to include the voltage layer called “myVertexHV” (see Figure 62).

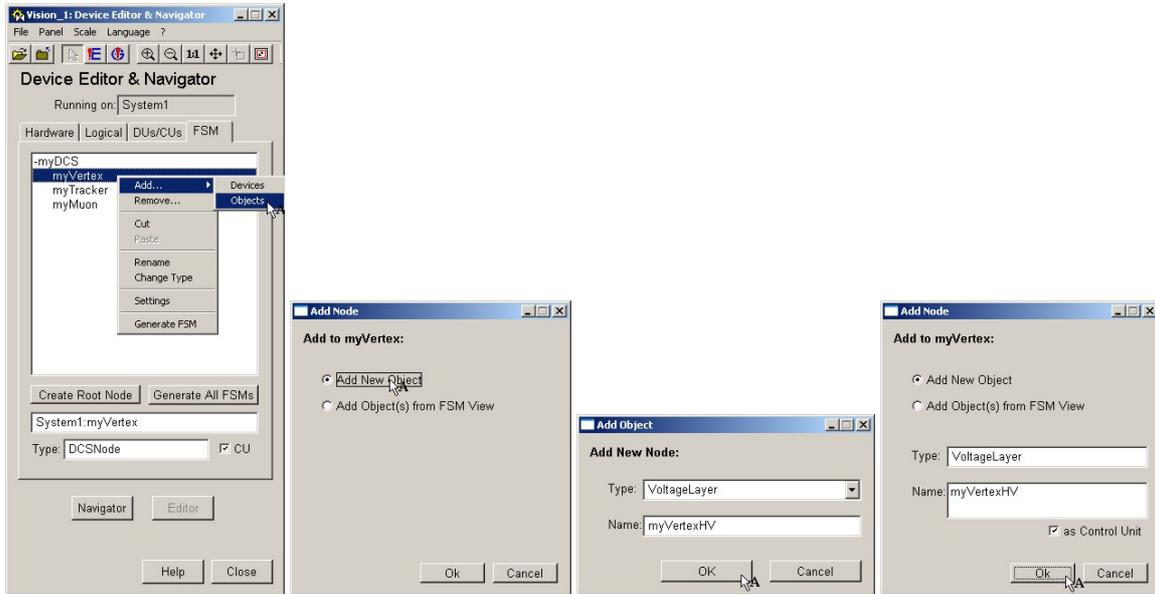


Figure 62 Steps to include the voltage layer for the vertex sub-detector

To this voltage layer, one can now attach voltage channel devices, e.g. of type *FwCaenChannelStandard*, as shown in Figure 63.

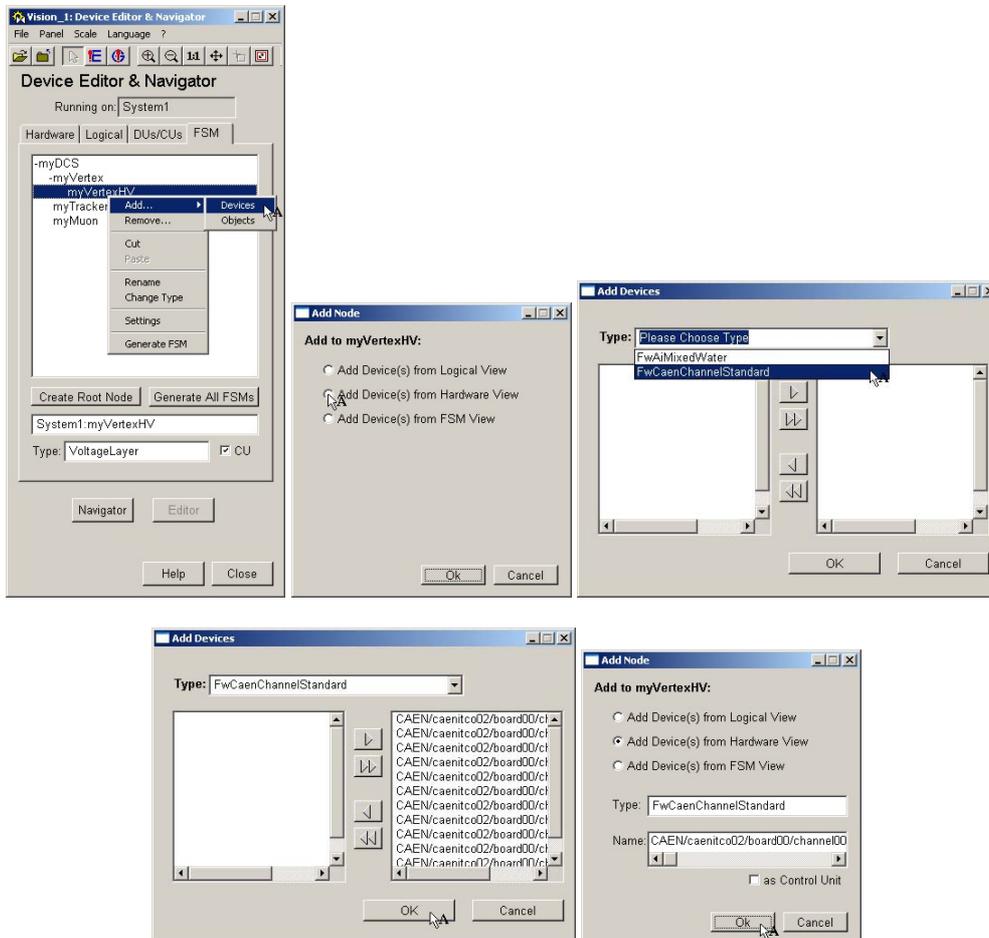


Figure 63 Steps to include voltage channels

Now this more complete hierarchy can be tested. For this, the tree has to be stopped and started again (see Figure 58) and after viewing the top node, one can branch down in the hierarchy by double-clicking on “myVertex” and subsequently “myVertexHV” (see Figure 64).

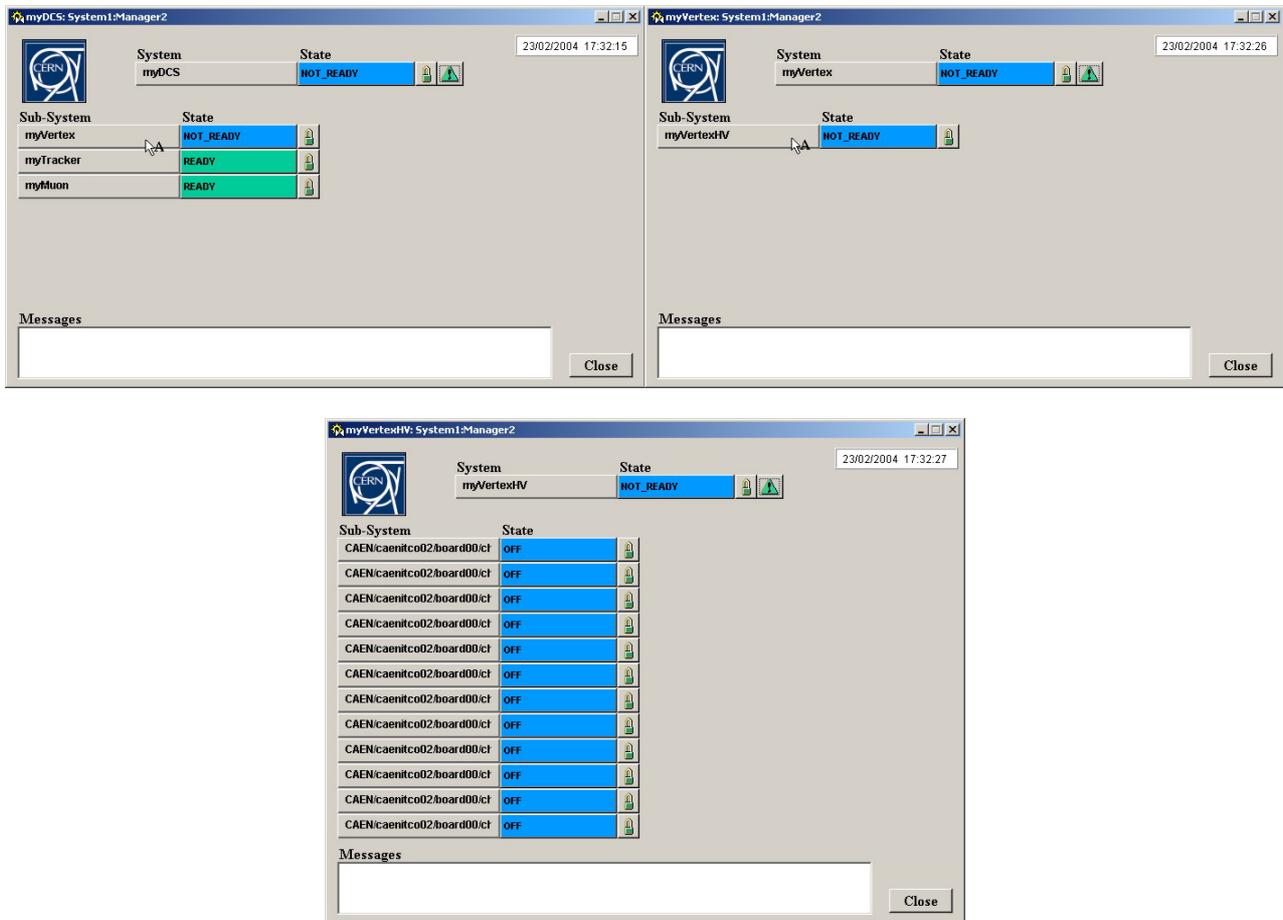


Figure 64 Branching through a running hierarchy

Again, all commands are available and can be tested. If a CAEN crate is connected or a simulator is used, also the channels will follow the commands.

The inclusion of the node “myVertexCooling” and the temperature sensor is left to the reader.

3.2.4. Trending in the JCOP Framework

The trending capabilities provided by PVSS are limited in contrast to those known to physics users. The JCOP Framework in its current version does not extend those capabilities, but provides a coherent way to configure trends, plots, and pages. As there are many terms around, it seems to be necessary to clarify them. This is tried in Figure 65. Note that “trend” is the general term for plots used in PVSS.

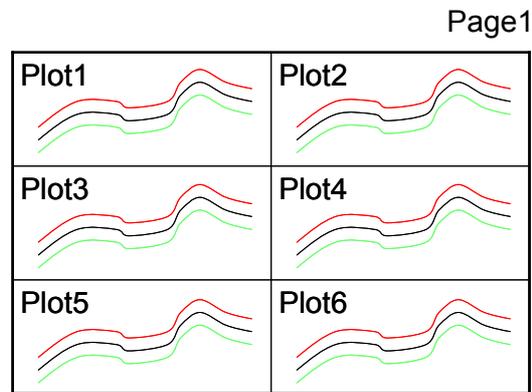


Figure 65 Sketch on pages and plots

In the current implementation, a page can have up to 6 plots, which in turn can have up to 8 curves. Plots and pages can – but do not have to – be organized in a tree structure. It is advisable to add a user interface manager for the trending main panel. Append a manager similarly to Figure 29 with the following options:

PVSS00NV –p fwTrending/fwTrendingMain.pnl

This will open the main trending tool panel as show in Figure 66. From this panel, which has a functionality similar to the DEN, the user can configure, view, and organize his plots.

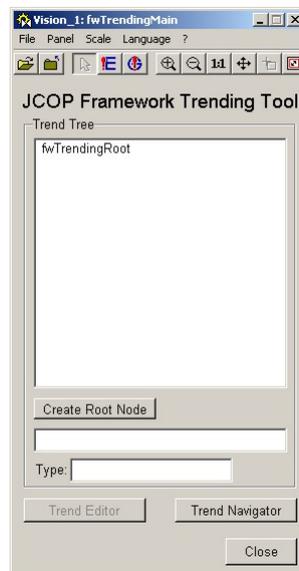


Figure 66 Main trending tool panel

In order to create new plots and pages, a right-click on any node in the trending tree, e.g. `fwTrendingRoot`, opens a context-menu from where one can select “View” in the trend editor mode only. The opening panel “Existing Pages, Plots, and Trending Tree Nodes” allows to perform all steps in connection with pages and plots. If one does not intend to use the trending tree to organise pages and plots, this panel provides all necessary functionality (see Figure 67).

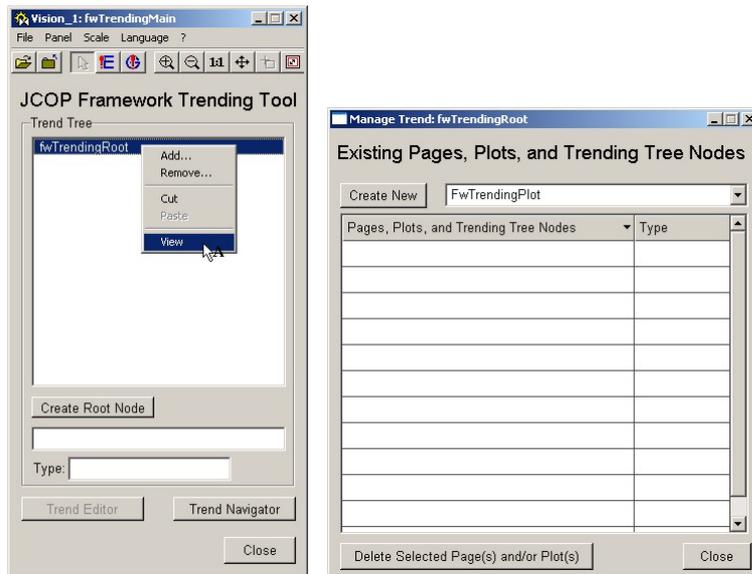


Figure 67 Handling of pages, plots, and trending tree nodes

At the top of the panel, one can create new pages, plots, or trending tree nodes by choosing the type and clicking on “Create New”. This will open a panel with the options for plots or pages.

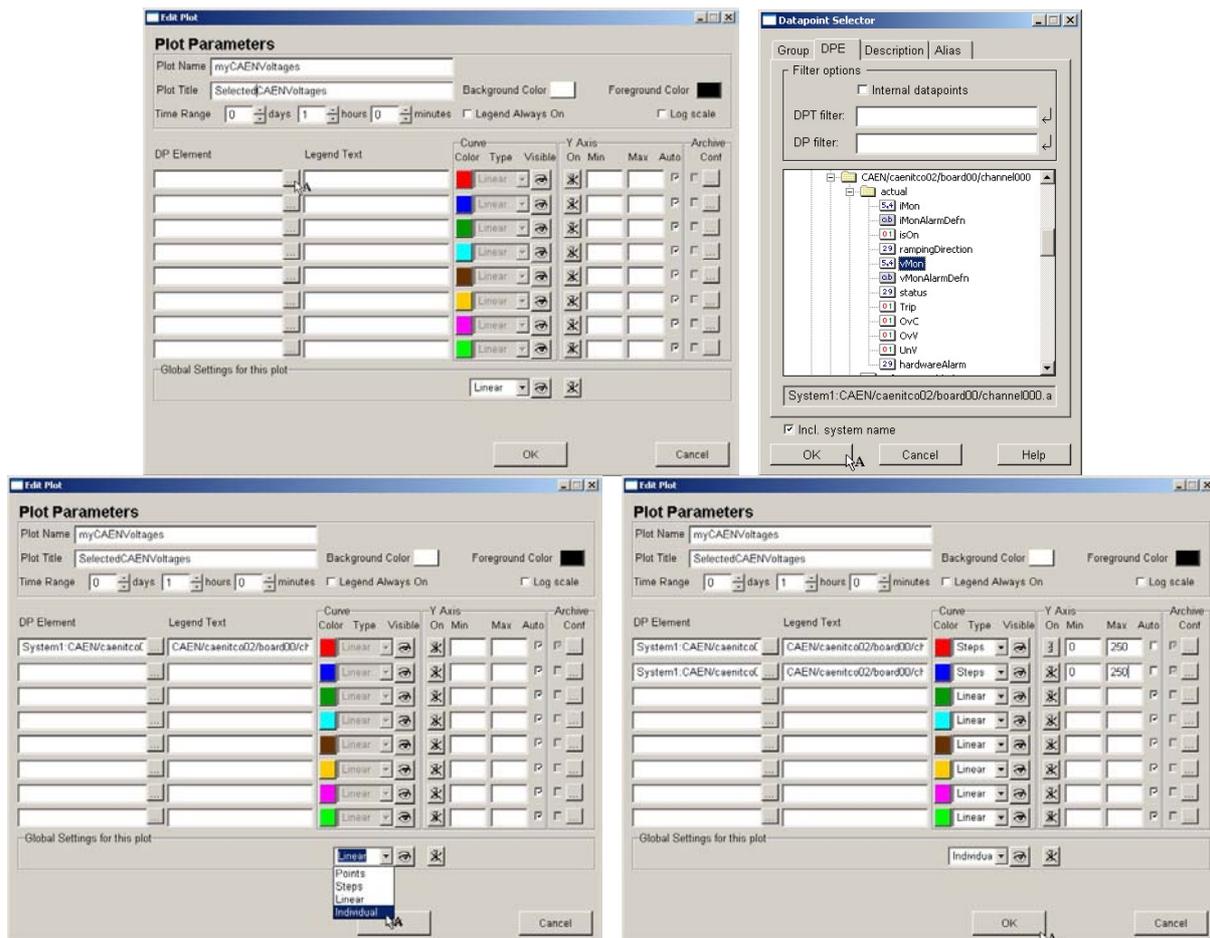


Figure 68 Basic steps to create a new plot

3.2.4.1. Creating a New Plot

The steps how to create a new plot are shown in Figure 68. The plot name needs to be unique, as this will be the name

of the description datapoint, whereas the plot title can be reused. Most of the settings available on the page are self-explanatory, the probably less obvious ones are:

- if one chooses a datapoint element for a curve, the element's alias is taken as curve legend by default but can be overwritten
- for curve types, curve visibilities, and y-axis visibilities, standard settings for all curves can be done using the switches at the bottom of the table (Note that one needs to set the curve type to "Individual" in order to be able to choose individual curve types

3.2.4.2. Creating a New Page

Similarly, one can create a new page (see Figure 69). The remarks on name and title for plots are also valid. Currently, pages can contain up to 2 columns with up to 3 rows. Setting the row and column numbers will create small synoptics, clicking on which one can set the plot to be displayed in the corresponding place.

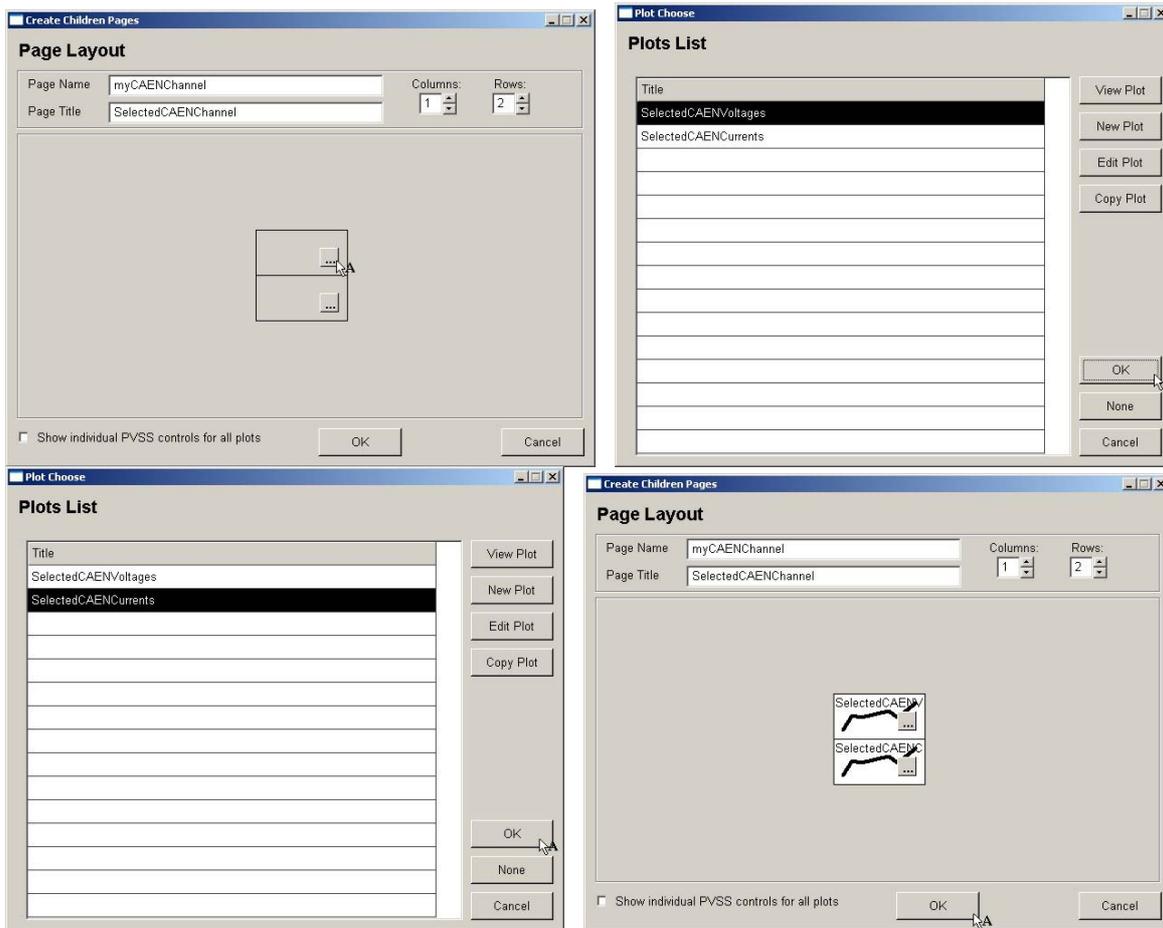


Figure 69 Basic steps to create a new page

3.2.4.3. Viewing Pages and Plots

Also from the same panel, one can change page or plot settings by double-clicking on their names, which will reopen the configuration panels (see 0 and 3.2.4.2). A right-click on a page or plot will display it. The following figures (Figure 70 and Figure 71) show some examples of plots and pages and the manipulation options.

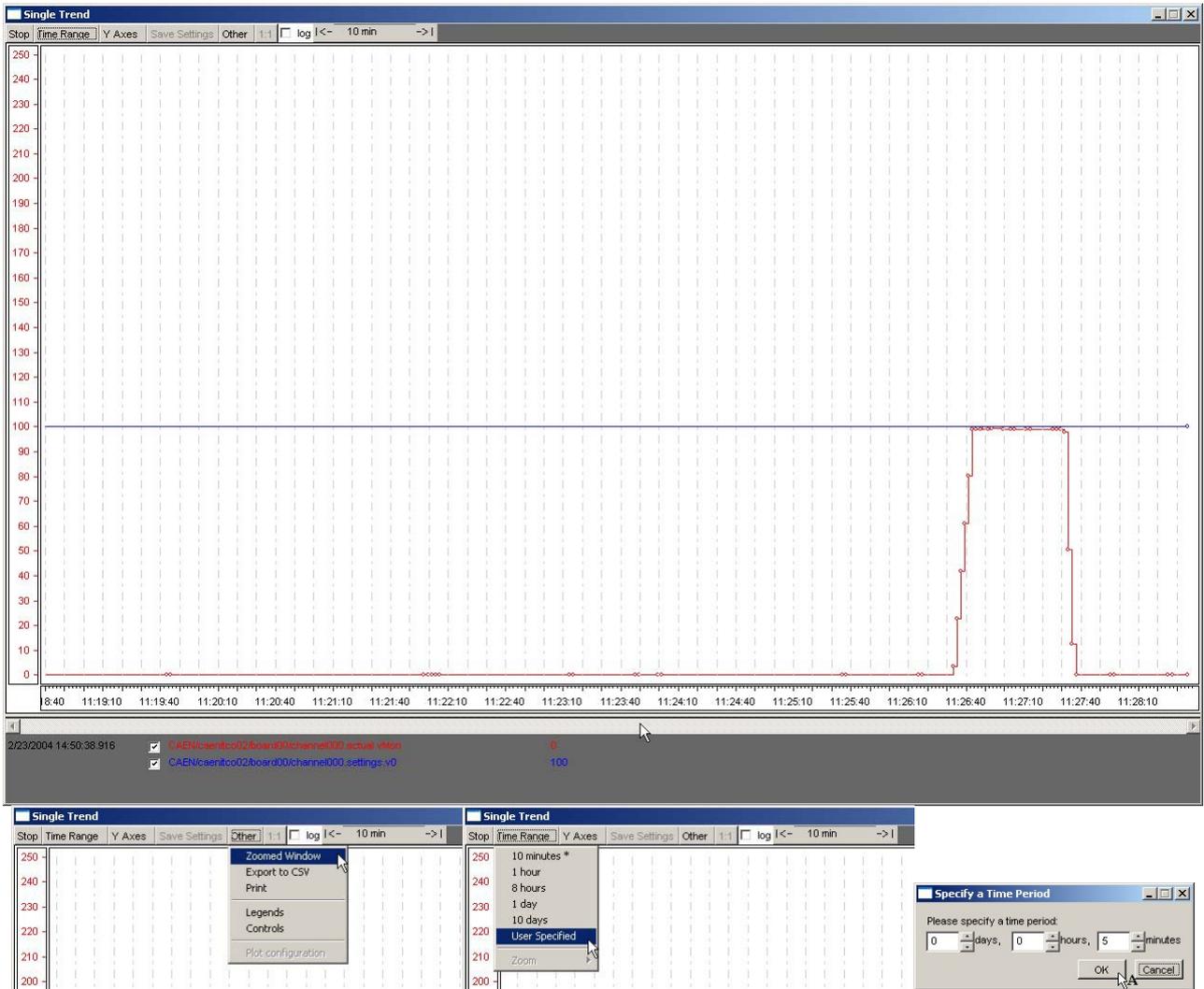


Figure 70 A single plot and some of the options

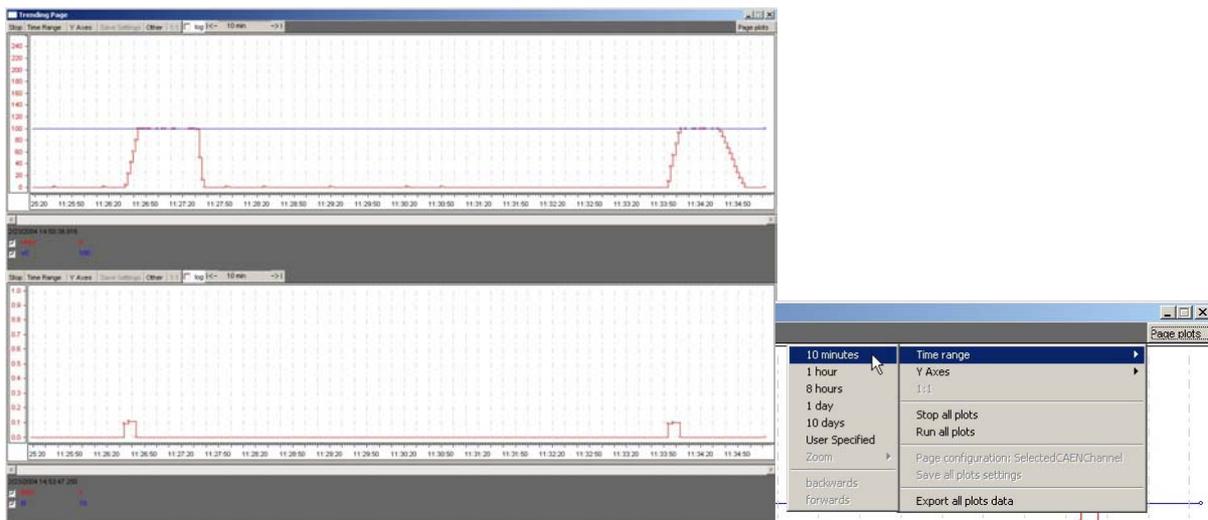


Figure 71 A page made of two plots and some page global options

3.2.4.4. Arranging Pages and Plots in the Trending Tree

The trending tree allows you to organize your pages and plots. This is for better retrieval only and does not influence the relations between pages and plots, and it does not allow to create pages or plots directly from the tree. The used tree widget is the same as the one of the FSM, so the same functionality exists (see Figure 72).

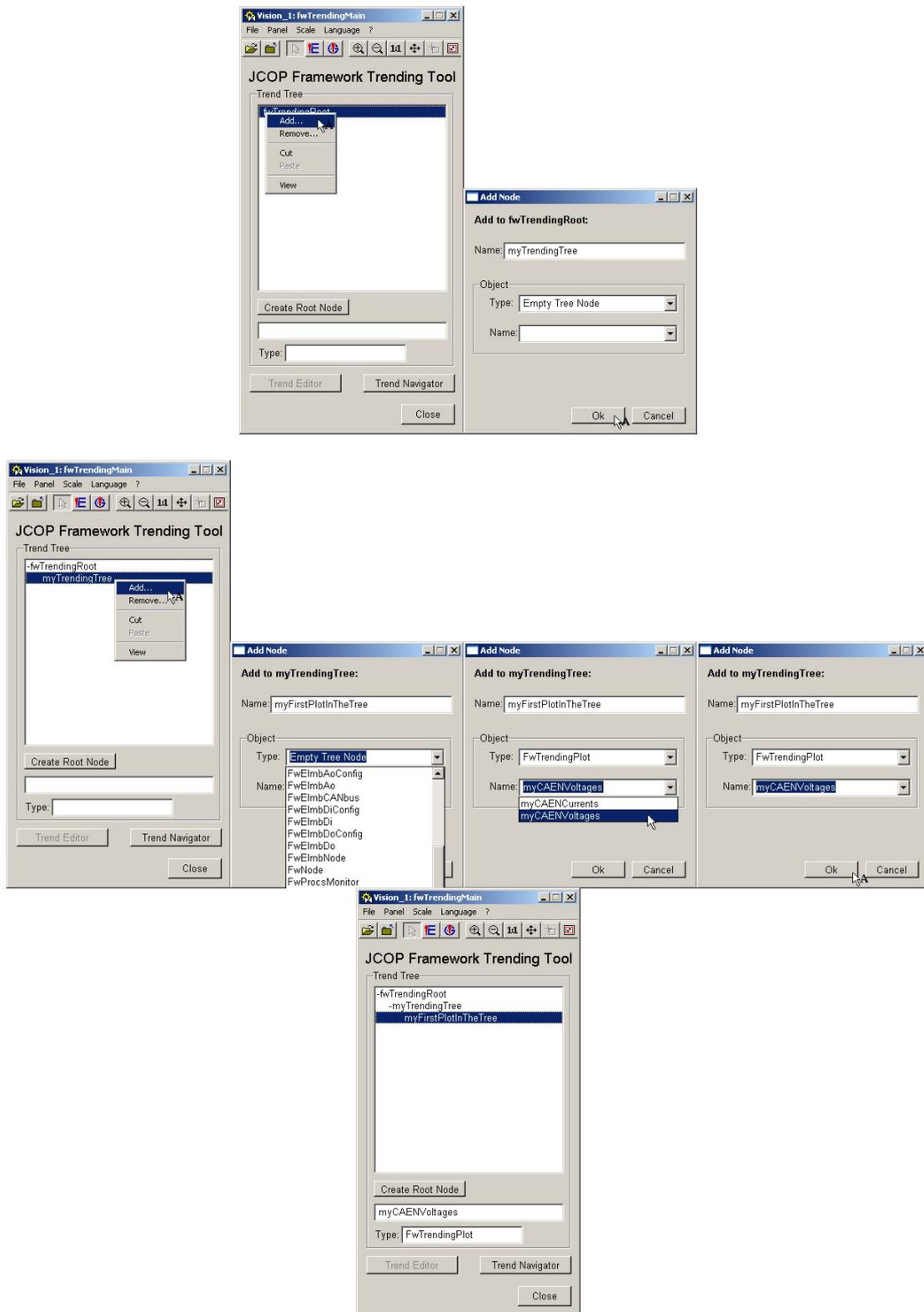


Figure 72 Steps to create a node and arrange a plot inside

3.3 User Interfaces based on the JCOP Framework

Coming soon

4. An Example Controls System

Coming soon

References

- [1] JCOP Architecture Working Group: “Framework Design Proposal,” CERN, Geneva, 2001, CERN-JCOP-2000-008, [online]. Available:
<http://cern.ch/itco/Projects-Services/JCOP/SubProjects/Architecture/pdf/AWGReport.pdf>
- [2] JCOP Framework Working Group: “Joint Controls Project (JCOP) Framework Sub-Project – Guidelines and Conventions,” CERN, Geneva, 2000, CERN-JCOP-2000-008, [online]. Available:
<http://cern.ch/itcobe/Projects/Framework/Documentation/guidelinesDocument.pdf>
- [3] Gisy, X.: “PVSSII Basic Course – Slides and Exercises,” ETM, Eisenstadt, 2003. Available on request from
ITControls.Support@cern.ch
- [4] Gaspar, C.: “Hierarchical Controls – Configuration & Operation,” CERN, Geneva, 2003, [online]. Available:
<http://cern.ch/clara/fw/FSMConfig.pdf>
- [5] Gaspar, C.: “Introduction to DIM,” CERN, Geneva, [online]. Available:
http://cern.ch/dim/dim_intro.html
- [6] Franek, B., Gaspar, C.: “SMI++ - State Machine Interface,” CERN, Geneva, [online]. Available:
<http://cern.ch/smi>