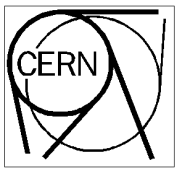


ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE
EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

Laboratoire Européen pour la Physique des Particules
European Laboratory for Particle Physics

Version: 1.3
Date: 20th October 2010
EDMS ID: 1100577

JOINT CONTROLS PROJECT (JCOP)
FRAMEWORK SUB-PROJECT
GUIDELINES
AND CONVENTIONS

Authors: JCOP Framework Team

Abstract

This document contains the Guideline and Conventions to be adopted when developing components of the LHC Experiment Control Systems as well as an overview of the components that will be provided as part of the Framework.

Table of Contents

1. INTRODUCTION

1.1. Purpose of the Document

This document contains the guidelines and recommendations for building control applications for the LHC Experiments. It forms one of the deliverables of the Framework Sub-project and the Framework[1] implementation itself also follows the guidelines and conventions laid down here.

1.2. Intended Audience

This document is intended to be used by the developers of the Framework as well as Control System Developers within each of the four LHC experiments. As such, although a deep understanding of PVSS is not required to read this document, some basic knowledge of Prozeß-Visualisierungs- und Steuerungssystem (PVSS) is necessary as standard PVSS terminology is used in many places. The PVSS specific terminology is highlighted by the use of Italics. The convention is also used where standard terminology might have a specific meaning in the context of PVSS.

1.3. Support

Support for the Framework and PVSS can be obtained from the IT/CO group via:

ITControls.Support@cern.ch

Further information on the Framework, including a download area, can be found under:

http://itcobe.web.cern.ch/itcobe/Projects/Framework/welcome.html

Further information on PVSS, including information on how to obtain a license as well as the software to download, can be found under:

http://itcobe.web.cern.ch/itcobe/Services/Pvss/welcome.html

Information on training courses for PVSS and the Framework can be found under:

http://itcobe.web.cern.ch/itcobe/Services/Pvss/Training/welcome.html

2. FRAMEWORK IN GENERAL

The section is intended to give an overview of what the Framework is intended to provide and what it is not intended to provide, i.e. what you as a user will be required to develop yourself and what you can expect to be provided to you. Of course this will not cover facilities that an experiment may decide to implement itself centrally and provide for all its users. A list of currently available FW components can be found under:

http://itcobe.web.cern.ch/itcobe/Projects/Framework/welcome.html

In general, the Framework is intended to provide all the standard features and facilities required by the four LHC experiments as originally described in the

[1] http://itcobe.web.cern.ch/itcobe/Projects/Framework/welcome.html

Architecture Document produced by the Architecture Working Group (AWG) and modified in discussions in the Framework Working Group (FWWG). In addition, it will address a number of 'Look and Feel' issues by providing standard symbols and template *panels* as well as a number of configured components e.g. for the configuration and operation of standard hardware devices.

Users of PVSS may request new features where they believe these will be of general interest to all experiments. Furthermore, if users develop features themselves which they believe to be of general interest they are encouraged to provide these to the Framework team to be considered for integration into the Framework. Such suggestions will be discussed within the FWWG.

Terminology:

A *PVSS application* which uses a single PVSS system (with a single *EV* and *DM*) but runs on multiple machines is termed a *Scattered System*. A *PVSS application* which is comprised of two or more *PVSS systems* communicating together is termed a *Distributed System*.

3. GENERAL GUIDELINES AND CONVENTIONS

3.1. Project Definition

When defining a new *project* the *project language* should be defined to be US English (*en_US.iso88591*). This is required in order to use PVSS provided *panels* which are supported in two languages; namely US English and Austrian German. Furthermore, all projects shall be developed as *distributed projects*, i.e. should have an assigned *System Name*, ideally be short, but should nonetheless be representative of the part of the overall control system that this *PVSS system* will be responsible for, e.g. atlasMuonEndCap.

Do not create *PVSS project*s or *PVSS project paths* which are longer than 64 characters.

3.2. Use of Standard *Panels* and *Scripts*

The Framework and ETM provided *Data Point Types* (*DPT*), *panels*, and *scripts* **shall not** be modified as this will lead to significant version control problems at a later date.

Should a modification be required to ETM or the Framework provided *DPTs*, *panels* or *scripts* than a change request should be submitted to the IT-CO PVSS Support Team via ITControls.Support@cern.ch

3.3. Panel Development

PVSS supports the possibility to instantiate generic *panels*, including inheritance of subsequent modifications. Therefore, it is recommended to use *reference panels* (with appropriate *$-parameters*) wherever possible. Even if the final design of such a reference panel is not known, this is not a problem. Create a simplified version of the generic *panel* and reference this *panel* where necessary. Once the design is known, modify this generic *panel* and all places where this *panel* is referenced will inherit these modifications.

Where *scripts* in *panels* execute identical or similar functions one should gather these functions into *libraries*. This will reduce the development effort, improve the re-usability as well as improving the maintainability of the application.

As a general rule, it is recommended that PVSS *panels* should display the displayState of an object, rather than the control knobs to change that state. As such it should be necessary to click on the object in some way to pop-up the appropriate control knobs using the standard mechanism defined by the Framework. This would reduce the risk of sending commands unintentionally.

When referring to a *Data Point Element (dpe)* within a *panel* or *script* then the *system name* should also be used as part of the *dpeName* in order that the same *panel* or *script* may be used within another *PVSS system* to access the same *dpe*.

3.4. Use of Control Managers

The *PVSS Console* entry *PVSS00ctrl -f pvss_scripts.lst* starts one *Ctrl Manager* with all the listed PVSS standard *scripts* started in a separate thread and therefore it is not possible to stop just one *script*. Stopping the *Ctrl Manager* would stop all the *scripts*, so if one needs to stop an individual script it is better to start a different C*trl Manager*.

Similarly, the *Console* entry *PVSS00ctrl -f fwScripts.lst* starts one *Ctrl Manager* with all the listed Framework standard *scripts*

3.5. Execution of *Control Scripts*

Controls scripts which are attached to *panels* run within the *UIM* into which the *panel* is loaded. Therefore, in a *scattered system* it is the machine used for the GUI which runs any *scripts* associated with the *panels* being viewed by the user.

If it is intended to reduce the load on the *UIM* and run these *scripts* on another machine then a difference approach is required. In this case the *script* should be written independently of the *panel* and run in a *Ctrl Manager*. Should this *script* need to react to user input then it must have a *dpConnect()* to one or more *dpes* that are modified via a *panel*.

4. GUIDELINES FOR DEVELOPMENT

4.1. Organisation of Folders

The organisation below refers to a single *project*.

4.1.1. Panels

PVSS uses relative references for *panel names*. The following folder structure for *project panels* is recommended:

For Framework *panels*:

- <Project>/panels/componentName – for *reference panels* which are used during the development of other *panels* (an example of a componentName would be fwWiener)

For application specific *panels*:

- <Project>/panels/app – for *reference panels* which are used during the development of other *panels*
- <Project>/panels/app/details – for dialog boxes, *panels* to modify parameters, etc. (pop-up *panels*).

For *symbols* and *catalogues*:

For Framework *symbols* and *catalogues:*

- <Project>/panels/objects/componentName – for basic symbols. Each of these subdirectories (e.g. fwWiener) would correspond to a catalogue set as shown in the *Native Gedi*. Only one level of hierarchy is possible.

For application specific *symbols* and *catalogues:*

- <Project>/panels/objects/app – for basic symbols.

4.1.2. Scripts and Script Libraries

It is recommended that the basic folder structure adopted by ETM is maintained wherever applicable. As such, the following folder structure for *project script* and *script libraries* is recommended:

- <Project>/scripts/componentName – for all scripts
- <Project>/scripts/libs/componentName – for all *scripts libraries*

4.1.3. Help

The help files should be stored in the following folder following the PVSS Tree structure afterwards:

- <Project>/help/en_US.iso88591/componentName/

For panels this would imply:

- <Project>/help/en_US.iso88591/componentName/panels/componentName/<panelName>.htm

For scripts this would imply:

- <Project>/help/en_US.iso88591/componentName/scripts/<scriptName>.htm

For script libraries this would imply:

- <Project>/help/en_US.iso88591/componentName/scripts/libs/<scriptLibraryName>.htm

The help files for panels should follow the structure given below. An example is given in Appendix ??. Further examples may be found under:

http://itcobe.web.cern.ch/itcobe/Projects/Framework/Documentation/

Panel Name:	e.g. *CaenChannelOperation.pnl*
Introduction:	A brief summary of the function of the panel.
Instructions:	A set of instructions on how to use the panel. If appropriate this may be a step-by-step guide. In the case where there are several options then all options should be addressed.
Restrictions:	Any restrictions on how the panel may be used.

Screenshot of Panel:	One, or more, screenshots of the panel. These should show all options.
Dollar Parameters:	The list of dollar parameters used in the panel, a description of each and an indication as to whether they are mandatory or optional.

4.1.4. Message Catalogues

The project specific message catalogues should be stored in the following folder:

- <Project>/msg/en_US.iso88591/<fileName>.cat

4.2. Version Control

It is recommended to store all developments in a central software repository. The repository can be organized in directories with the different components. Every developer has his own PVSS project with all the managers he needs. One of the project paths is set to the last release generated from the repository.

When he wants to change one of the components in the repository, he checks out the component to his local file system, adds it to the project path and then does the necessary modifications. Since changes can only be saved in the first project directory, afterwards he will have to move by hand the changed files to the directory where the component was checkout. When he has finished, he checks in the component.

These changes will not be seen automatically by all the developers unless they check out the same component. Periodically, new releases are generated from the repository for all the developers. This is done by a Configuration Manager. He is in charge of keeping the coherency among all the developments.

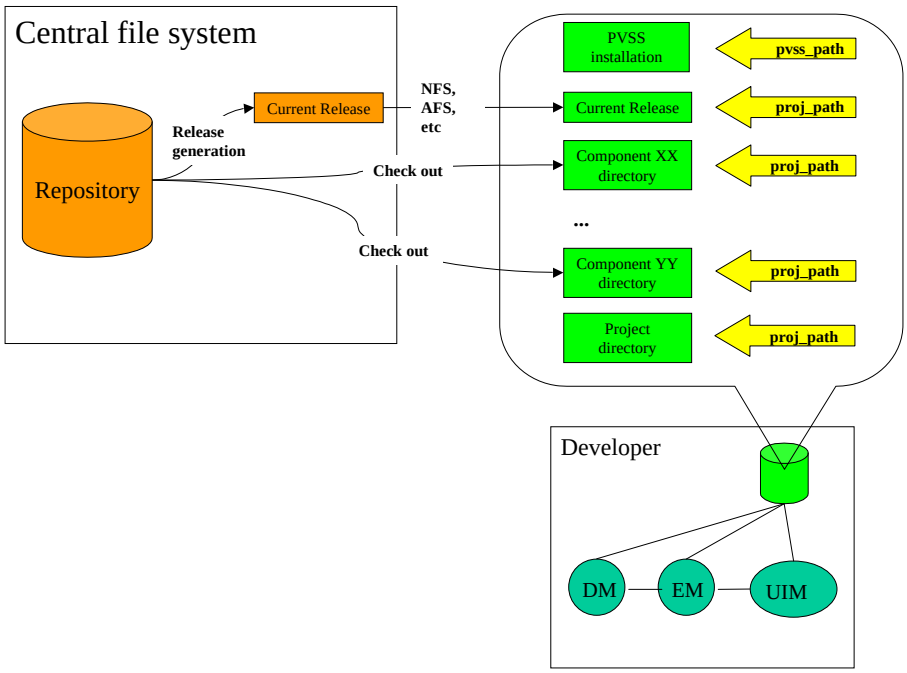


Figure 1. Development with a central repository.

Advantages:

- Allows to track changes,
- Provides control of concurrent access
- Allows easy separation of development into components
- Easily scalable to a big team of developers

The recommended way for development is to use a software repository. This repository can be set with the different available tools: CVS, Razor, etc.

5. NAMING CONVENTIONS

5.1. General

It is recommended that the PVSS standards should be adopted whenever there is no good reason to deviate from these. In this way consistency will be maintained between *PVSS system* definitions, Framework definitions and user definitions. As a result the interCapNotation should be followed wherever possible. The interCapNotation requires that a name, which is comprised of multiple words, is written as a single word in which the start of each new word is written with a capital letter and all others in small letters. With the exception of *DPT* names, all names should start with a small letter.

Below are a number of additional general conventions:

- The underscore "_" is reserved for special usage and therefore its use should be avoided except where needed to follow the recommendations.
- PVSS supports long names so advantage should be taken of this and the use of abbreviations should be avoided. Hence, meaningful names should always be used.
- The names should be of sufficient length as to reduce the risk of duplication. Therefore the *panel* for a generic tree browser would be better called fwGenericTreeBrowser.pnl than either fwTree.pnl or fwBrowser.pnl as these names could easily refer to other things.
- Every *project* should be created as a *distributed system* and the *system name* should always be used when referring to *data points*. For generic *panels* the *system name* should be passed as a *$-parameter* as part of the *data point name*.
- The names should include a differentiation for system and framework entities from application specific entities. For framework entities the prefix fw has been chosen. Therefore, non-framework entities must not start with this letter combination.

5.1.1. PVSS System Names

Each *PVSS system* has a system name which must be defined in order to be able to link two *PVSS systems* together. By default the name dist_systemNumber is given to a *PVSS System*. *The system name* is used as part of the *data point names* as given below:

[systemName:]dpName.[dpElement(s)]:config.[detail].attribute

- *systemName*: name of the *PVSS system*
- *dpName*: name of the *data point*
- *dpElement*: *element name* of the *DP type*

- *config*: *attribute group*
- *detail*: *detail number* of the *attribute group*
- *attribute*: real value of an *attribute*

In order for two *PVSS system* to communicate the *system names* and *system numbers* must be different. Therefore, a convention is required within an experiment (or other domain) to ensure that the *system names* and *system numbers* are unique within that experiment (domain).

5.2. Naming conventions

Below is a list of specific naming conventions:

- *PVSS Systems* – All *PVSS projects* should include a *system name* which should be the name of the logical entity being covered, e.g. muon or muonHv. This should be kept relatively short, but still meaningful, as this will be used in all *DP* names.
- *Data Point Types (DPT)* – *DPT* names should start with a capital letter. For framework *DPTs* the following convention should be adopted FwDeviceName and for non-framework *DPTs* DeviceName, *e.g.* MyWienerChannel. For internal *DPTs* a preceding underscore is used, e.g. for FW internal *DPTs* _FwInternal (_FwDeviceEditorNavigator) *DPT* and for non-FW internal *DPTs* _InternalDPT.
- Device Instances – a *data point* should begin with a small letter. The recommendation is that the name should be based on the hardware view of the device, i.e. its hardware connection, e.g. rack053Crate05Board06Channel03. The *dpAlias* should be its position in the logical hierarchy with each level separated by the foreward slash "/", e.g. muon/EndCap/HV/Wire064.
- *Panels/Libraries* – *panels* should have meaningful names and if several *panels* belong to the same device/subsystem then they should have a prefix related to the device/subsystem and combined with a description of the specific usage e.g. muonGeneralOverview or muonState, etc. Alternatively, where a *panel* is used to display information from a specific *data point type* then the name could be based on the *DPT* name e.g. fwCaenChannel.pnl (for the *DPT* fwCaenChannel).
- Recipes – the name should refer to both the grouping included and the operational conditions being modelled e.g. muonNominalState, muonCalibration, etc.
- *Scripts/Libraries* – *script libraries* should either have the name of the *DPT* for which they contain the behaviour e.g. fwCaenChannelConfiguration.ctl (for the *DPT* fwCaenChannel) or for general *libraries* a name describing the group of functions provided e.g. a *library* to manipulate *peripheral address* could be called fwPeriphAddress.ctl. *Functions* within a *library* should include the *library* name (followed by an underscore) within the *function* name, e.g. fwPeriphAddress _set(). Alternatively, if this produces *function* names which are too long instead a library name could be preceded with a prefix which is then used for all *functions* within, it, e.g for the above example the *library* could be called pa_fwPeriphAddress.ctl and the function pa_set().

  Framework *libraries* should commence with fw e.g. fwAlertConfig.ctl. *Functions* which are private to a *library* (whether a Framework or user specific library), i.e. are only used within the *library* to simplify other *functions* and which should not

be available directly to the user, should commence with an underscore (_), e.g. _fwAlertConfig_createAnalog2(). *Function* names must include a verb.

- *Callback (work) functions* should have a postfix of *CB* so that these *callback functions* are easily identifiable e.g. calculateAverageCB()
- Global variables – begin global variables with g_ (i.e. g followed by an underscore), e.g. g_threadId.
- Variables with defined meaning:

  When the meaning of a variable needs to be identified via the name then the above prefixes should be used e.g. dpTest1 or dpeValve

dp	data point
dpe	data point element
dpc	data point configuration
dpa	data point attribute
dpt	data point type

- Use of *$-parameters* – to ease the understanding of the information contained in *$-parameters* the following prefixes are recommended for the naming:

$dp, $dpe, $dpa, $dpt	as above
$i	integer
$f	float
$s	string
$u	unsigned
$c	character
$b	boolean
$aPvssDataType	arrays of the any of the PVSS standard data types e.g. $as for a dynamic array of strings, $ai for a dynamic array of integers, etc.
$ti	absolute time (YYYY.MM.DD HH:MM:SS.mmm)
$tp	time period in seconds
$td	date (YYYY.MM.DD)
$tt	time (HH:MM:SS.mmm)

  Example: $dpEngine, $iMin, $sValveName, $bIsValid, $tiStart, $tpDuration, $ttStartToday, $sErrorStateColour, $sPopUpPanelName
- States – should be simple and clearly understandable e.g. On, Off, Standby, etc. and should be taken from a default list (tbd) if applicable – awaiting output from AWG. An initial set might be:

ON
OFF
RAMPING UP
RAMPING DOWN
ERROR
PHYSICS
CALIBRATION

STANDBY
INITIALISING

- Actions – should be simple a verb and be clearly understandable e.g. Switch On, Switch Off, Go To Standby, etc. and should be taken from a default list (tbd) if applicable – awaiting output from AWG. An initial set might be:

SWITCH ON
SWITCH OFF
RAMP UP
RAMP DOWN
GOTO PHYSICS
GOTO CALIBRATION
GOTO STANDBY
INITIALISE

- File names – these should follow the standard interCapNotation e.g. fileName.ext
- *Data point configurations* – should always use the *language independent* representation e.g. dpeEngine + ":_original.._value"

6. LOOK AND FEEL CONVENTIONS

6.1. General

Where standard PVSS *panels* exist, e.g. for *alarm* display and trend displays, it is recommended to use these. If required, modified versions of these will be provided with later versions of the Framework.

6.2. Panels

6.2.1. Panel Sizes

Screens with at least a resolution of 1280/1024 are recommended.

The standard *panel* sizes should be:

- Standard *panel*: 1269/894 pixels
- *Child panel*: 1126/724 pixels

However, should this resolution prove too difficult to read on small screens then a minimum resolution of 1024x768 would be recommended. The maximum recommended *panel* sizes for this resolution would be:

- Standard *panel*: 1013/650 pixels
- *Child panel*: 870/580 pixels

6.3. Colours

A palette of colours is provided with the Framework which overrides the ETM standard palette to support the colour conventions given below i.e. defined colours with appropriate names. It is important that the named colours are used, i.e. the name and not the RGB representation.

The Framework will support the standard ETM default colours where applicable but will give them more appropriate names.

6.3.1. Colours to be used when developing panels

- Background colours

Default in *GEDI*	***_3DFace***[2]
Default in *panel attribute*	***_3DFace***
Panels, objects (*button, radio box, check box, table, cascade, text field and list*)	***_3DFace***
Text fields and *lists* when these are input fields	***_Window***[3]

- Foreground colours

Default in *GEDI*	***_3DText***[4]
Object in *alarm* state should display the *alarm colour* but the foreground colour should be	***_3DText***
Text fields and *lists* when these are input fields	***_WindowText***[5]

- Corresponding to equipment status (on/off, enabled/disabled, *alarm* masking, in *alarm* ...)
  - ♦ State information given with text e.g. ON, OFF or the value given e.g. 300 V and optionally the background colour is set as below:

FwStateOKPhysics	Green	Okay and in physics data taking
FwStateOKNotPhysics	Blue	Okay but not in physics data taking
	'*Alarm* colour'	The colour corresponding to the alarm status as given below
1) FwBackground 2)FwEquipmentDisabled	Background set to the colour of the background (1) and text greyed out but the colour should be named (2) to allow this to be changed independently of the background colour if necessary	Equipment Disabled
1) FwBackground 2)FwAlarmMasked	Background set to the colour of the background (1) but the text remains normal but	*Alarm* Masked

[2] Corresponding to ETM's _3DFace
[3] Corresponding to ETM's _Window
[4] Corresponding to ETM's _3DText
[5] Corresponding to ETM's _WindowText

	the colour should be named (2) to allow this to be changed independently of the background colour if necessary	
FwDead	Grey	Equipment not reachable or data invalid

- Miscellaneous
  - *Data point* does not exist – purple (***DpDoesNotExist***)

6.3.2. Colours used with Framework-provided Components

- Control modes of CUs or DUs:

FwModeIncluded	Green	Included
FwModeOther	Blue	Excluded, delegated, ignored or disabled
FwModeTreeIncomplete	Yellow border	Indicating that a some lower level node in the hierarchy is not in the Included mode
	Greyed out	From the list of mode actions those which are not available (PVSS standard disable mechanism)

- FSM States of CUs or DUs:

FwStateOKPhysics	Green	Okay and in physics data taking
FwStateOKPhysics	Blue	Okay but not in physics data taking
FwStateAttention1	Yellow	First level of error severity[6]
FwStateAttention2	Orange	Second level of error severity
FwStateAttention3	Red	Third level of error severity
	Greyed out	From the list of actions those which are not available (PVSS standard disable mechanism)

- *Alarm screen* colours:

FwAlarmFatalUnack	Red	Fatal/Came/Unacknowledged
FwAlarmFatalAck	Red	Fatal/Came/Acknowledged
FwAlarmErrorUnack	Orange	Error/Came/Unacknowledged
FwAlarmErrorAck	Orange	Error/Came/Acknowledged
FwAlarmWarnUnack	Yellow	Warning/Came/Unacknowledged
FwAlarmWarnAck	Yellow	Warning/Came/Acknowledged
FwAlarmFatalWentUnack	Grey	Fatal/Went/Unacknowledged
FwAlarmErrorWentUnack	Grey	Error/Went/Unacknowledged
FwAlarmWarningWentUnack	Grey	Warning/Went/Unacknowledged

[6] It will be the responsibility of the experiments to define which level of severity a particular CU/DU state should correspond to (level 1 is the lowest level)

6.4. Fonts

For fonts ETM recommend Arial (MS windows) and Courier New (MS windows) for NT and Helvetica (Adobe) and Courier (Adobe) for Unix. The default font is arial size 10 for NT.

It is recommended to do all development with the MS fonts as there is a config option under the UI section to perform a mapping between NT fonts and Unix fonts (see Style-Guide für PVSS-II-Projekte, page 35), but not in the opposite direction. An example of this mapping is provided with the Framework.

In general bold fonts should be used sparingly.

The following default font sizes should be used:

- Description for an entry field: Arial regular 10
- Text field: Arial regular 10
- Number field: Arial regular 10
- Title: Arial regular 14

Descriptive text, titles, etc. should always start with a capital letter and be followed by the remaining text in small letters.

To set the default font and font size in the PVSS00NG do the following:

- Open a panel
- Select View-Toolbars-Style from the menu
- Select the font, the size, etc.

These settings are kept and are valid for all the PVSS00NG of your project.

6.5. Buttons

Descriptive text should always start with a capital letter and be followed by the remaining text in small letters. Panel titles should be in CAPITALS.

If a command/action associated with a button is not allowed at this time then the button should be *disabled*. In the case where a button is used to select a pull down menu of options then the button should remain active and the menu items not available at that time should be greyed out.

- Standard button labels:
  - Ok
  - Cancel
  - Apply
  - Close
- Where the first letter is uppercase and everything else is lower case except for “Ok”.
  - “Cancel” means “I have changed my mind and therefore I want to return without implementing any change”
  - “Apply” means “apply the changes but don’t close the window yet after which all modifications to this point can no-longer be cancelled”
  - “Close” means “close this window without making further changes. Modifications that have already been applied still stand”
  - “Ok” means “apply anything that hasn’t already been applied and then close”

6.6. Panel Documentation

For each panel there should be an associated html file.

The file should be located in a directory inside help/en_US.iso88591/. For example, if we have a panel called fwDeviceCreate.pnl, located in panels/fwDevice and it belongs to the Framework component fwDevice, then the reference file fwDeviceCreate.html should be located in help/en_US.iso88591/fwDevice/panels/fwDevice/. So the general path form for a Framework panel is help/en_US.iso88591/fwComponentName/panels/fwComponentName/

The structure for documentation related to applications is similar substituting fwComponentName by ApplicationName.

6.7. Script Documentation

Template from Manuel.

7. PROGRAMMING CONVENTIONS

7.1. Control Script Conventions

The following is a list of conventions to be followed when developing control scripts.

- main() shall be declared first.
- Functions shall be separated by // ----------
- Each function shall include the following comments in a header as a minimum:
  - Name of the author
  - Date of last modification
  - Version number
  - Purpose of the function
  - List of input parameters and their meanings
  - List of output parameters and their meanings as well as any return value
- In addition, each library should include a header giving the purpose of the *library,* the functions provided by it and the version number.
- A magic number is any number other than 0 or 1. Programs should contain NO magic numbers but rather should use names. When trying to maintain a program it is very difficult to understand the meaning of magic numbers. However in PVSSII, CONST is not available, nor is #define, so use global functions in a standard library co_Constant which returns the value e.g. co_VALUEMAX() where the function name itself should be in CAPITALS.

7.2. Control Script Exception Handling

All control script code should include an exception handling mechanism which should be based on the standard Framework implementation which is described in Appendix B – Exception Handling.

7.3. *Alarm* Handling

Following the guidelines of the AWG the intention of an *alarm* is to bring an anomaly situation to the attention of an operator and as such *alarm*s are considered to be messages which are displayed to the operator via the *alarm display* and that are logged. An *alarm* does not initiate an action. Should an action be required then this should be handled within the FSM.

An *alarm* has several properties:

- Its ***Origin***, which is used to identify the source of an *alarm*.
- Whether or not the *alarm* requires ***acknowledgement.***
- Its ***Severity Level***, which is used to characterise the seriousness of the *alarm*.
- Its ***Dependencies*** on other *alarm*s.
- Certain ***Additional Details*** about the *alarm*.

7.3.1. Standard Alarms

These are *alarms* which are attached directly to *data point elements* using the PVSS *Alert Handling config*. There are two types which are proposed by the AWG and for which support is provided directly by the Framework.

These are the simple *boolean alarm* (*good value* and *alarm value*) and *analogue alarms* with up to a maximum of five *ranges* as shown in an example below. However, it is also possible to attach an *analogue alarm* with 2, 3 or 4 *ranges* as well. The Framework provides a mechanism to select the number of *ranges* desired. An example of an analogue parameter with five *alarm ranges* is shown below.

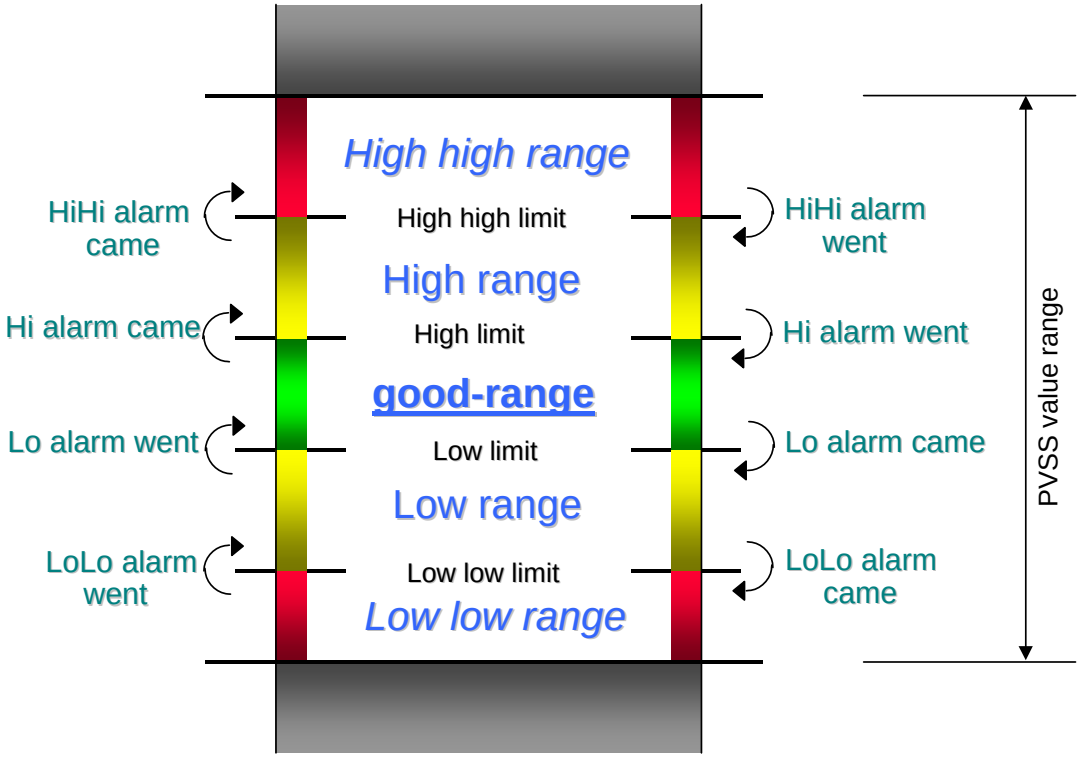


Figure 2: Analogue Alarm Handling with 5 Ranges

Alarms should be used whenever the excursion of a parameter from its nominal range of values needs to be brought to the attention of the operator. In normal circumstances an *alert handling config* would be attached directly to the parameter of interest (*dpe*). However, if the *alarm* condition were dependent on multiple parameters then it is

necessary to create an additional *dpe* with a *dpFunction config* as well as an *alert handling config*. This *dpFunction* would be written to reflect the *alarm* condition and to return a boolean value on which the PVSS alert handling would act.

The Framework will define *Alarm Classes* for the three levels of severity:

- Warning (W – will be used as the *short sign* for *alarms* of this *class*)
- Error (E – will be used as the *short sign* for *alarms* of this *class*)
- Fatal (F – will be used as the *short sign* for *alarms* of this *class*)

Two *acknowledgement schemes* are supported by the Framework. Firstly, as shown below:

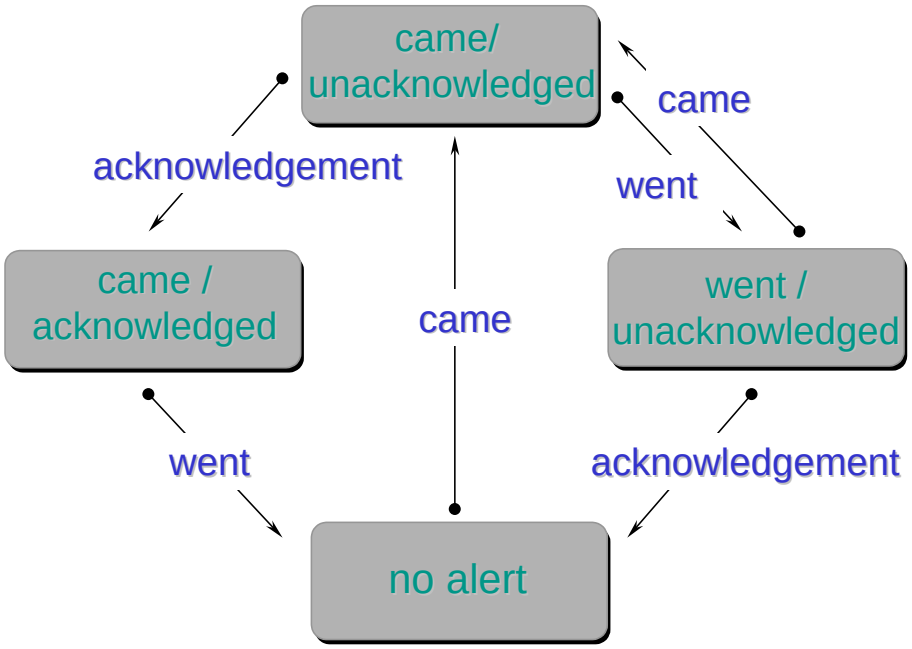


Figure 3: Alarm Acknowledgement Concept

In the above diagram the *alarm* is required to be *acknowledged* before it disappears from the *alarm* screen. Should the *alarm* become *inactive* before it is *acknowledged* and then become *active* again and only one *acknowledgement* is required for both instances of the *alarm*. The Framework will also allow for *alarms* that do not require *acknowledgement* i.e. they disappear from the *alarm screen* as soon as the *alarm* becomes *inactive*. The following *alarm classes* will be provided by the Framework:

- _fwWarningAck
- _fwWarningNack
- _fwErrorAck
- _fwErrorNack
- _fwFatalAck

7.3.2. Summary Alarms

PVSS *Summary Alarms* may be used, as the name suggests, to summarise *alarm* situations to obtain a level of abstraction. A *Summary Alarm* can be defined for any group of *alarms*. When active, a *Summary Alarm* indicates that at least one of the *alarms* in that group is active. It can be used to provide an abstraction for higher levels in the controls hierarchy e.g. at the sub-detector level to indicate that there is an *alarm* in a subsystem of that sub-detector. Alternatively, a *Summary Alarm* could be

used to group *alarm*s across multiple branches of the hierarchy e.g. each sub-detector may have a LV system and one might wish to have a *Summary Alarm* that groups all LV *alarm*s independent of the controls hierarchy i.e. to indicate that there is a LV *alarm* someone in the detector.

When defining a *Summary Alarm* one can use a *dp-list* or a *dp-pattern*. For performance reasons it is recommended to use a *dp-list* rather than a *dp-pattern*.

7.4. Access Control

To be replaced by new concept.

8. DEVICE MODELLING

8.1.1. Device Definition

To be provided by Manuel.

9. APPENDIX A – EXCEPTION HANDLING

This Appendix explains the method of handling exceptions within *panel* code and *library functions*.

Firstly, the requirements for writing the *library functions*:

All library functions should be written with a parameter to handle exceptions[7]. This parameter should be a reference to a *dyn_string* and should be the last parameter in the list of function parameters.

e.g.,

```
myFunction( ... , ... , dyn_string &exceptionInfo)
{
}
```

Exceptions can be tested for manually, for example testing if a value is outside a valid range. However, the failure of internal *PVSS functions* should also cause exceptions. Errors in the operation of most *PVSS functions* can be detected using the *PVSS function* called *getLastError()*. This returns a *dyn_errClass*. If the length of this is greater than 0, then a *PVSS function* has failed and an exception should be raised. If this is the case, then the *PVSS function throwError()* should be used, followed by the function required to raise the exception. This is explained below.

e.g.,

```
dyn_errClass err;

dpSetWait(dpe+":_smooth._type", DPCONFIG_NONE);
err = getLastError();

if(dynlen(err) > 0) {
   throwError(err);
```

[7] This parameter *may* be omitted if the function does not produce any error message/code. However, a function, which needs to return or pass an error code or message, should use this "exception" method exclusively.

```
   fwException_raise(exceptionInfo, "ERROR", fwSmoothing_delete():
   Could not delete config", "-1");
}
```

e.g.,

```
if(minValue >= maxValue)
{
       fwException_raise(exceptionInfo, "ERROR", "fwPvRange_set():
       Invalid range", "3");
       return;
}
```

NOTE: The use of the *throwError()* function is only required when the *getLastError()* function was used.

As shown above, if an exception needs to be raised, the function fwException_raise() should be called. This function can be invoked as shown:

```
fwException_raise(exceptionInfo, exceptionType,
                     exceptionText, exceptionCode);
```

- exceptionInfo is the dyn_string which contains the information about all current exceptions.
- exceptionType is a string containing the type of exception
- exceptionText is a string containing the description of the exception
- exceptionCode is a string containing a code associated with the exception

This function appends three items to the *dyn_string* which contains the exception information.

The types of exception which can be used are "ERROR", "WARNING" and "FATAL" depending on the severity of the exception.

The description of the exception should be a string in the following format:

```
"functionName(): details of exception"
```

Once an exception has been raised, it is likely that the rest of the function should not be executed. So a check should be inserted where necessary to see if any exceptions have been raised and if so, the function should return.

e.g.,

```
if(dynlen(exceptionInfo)>0)
       return;
```

This should be used after any part of a *library function* which can raise an exception. Also it should be used after any call to a function which uses the exception *dyn_string*. In this way, even if many functions have been called from within one another, they will all return and execution of the code will be stopped. Once the functions have all returned, the developer then can handle the exception in the *panel* which first called the *library function*.

Now, the requirements for writing the *panel* code:

Before a function call to a function which uses the exception *dyn_string*, it is necessary to create the exception *dyn_str*ing. This can then be passed to the functions.

If a function raises an exception, it will return after having placed the exception in the *dyn_string*. The developer must now decide how to handle the exception. Firstly, it is best to check if an exception has been raised at all. This can be done with the code...

```
if(dynlen(exceptionInfo)>0)
{
	//handle exception here
}
```

The *dyn_string* contains sets of three entries for each unhandled exception. The first part of the entry for each exception contains the exception type. The second part contains the exception details. The third part contains the exception code.

e.g.,

If two exceptions have been raised and neither have been handled yet, the *dyn_string* might contain:

```
["WARNING",
 "myFunction(): Value is out of valid range",
 "1",
 "ERROR",
 "myFunction(): Data point does not exist",
 "7"]
```

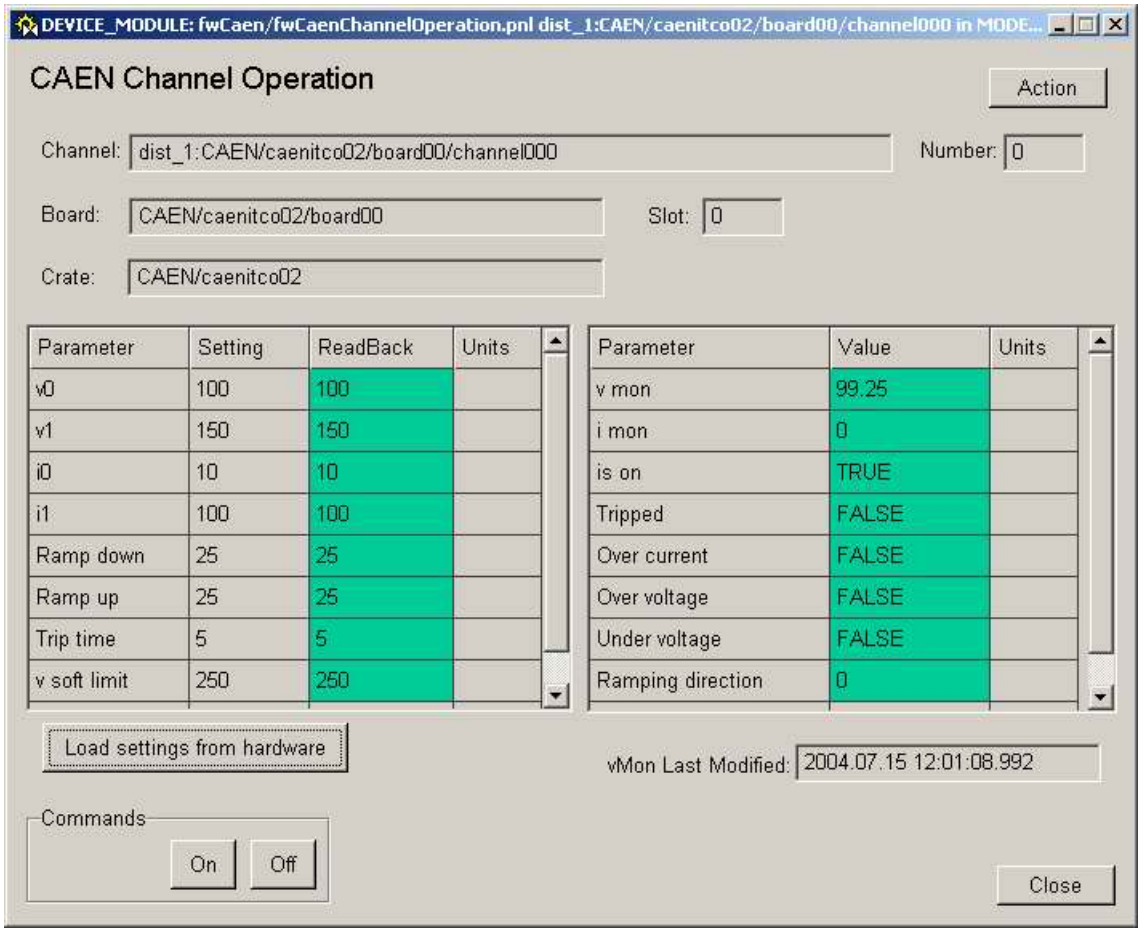
Two functions are available as standard to handle exceptions. One opens a new *panel* (fwExceptionDisplay.pnl) which allows the user to browse through all current exceptions. The other appends all current exceptions to a list in a given *PVSS list widget*. Both functions empty the exception *dyn_string* once they have handled the exceptions.

If the developer chooses another method to handle the exceptions, it is important to empty the *dyn_string* after all the exceptions have been handled.

10. APPENDIX B – PANEL HELP

Below is an example of a help file for a panel, in this case the CAEN channel operations panel.

Panel Name	fwCaenChannelOperation.pnl
Introduction	This panel is used to do the basic operations that can be performed on a CAEN channel.
Instructions	The left table represents the requested parameters that can be set for a CAEN channel, and the right table the actual values read from the channel. In the left table, the column Settting contains the values sent to the hardware. The column ReadBack contains the values that are actually stored in the hardware. It is possible to ask for a different value by clicking on the column Setting and the row of the paremeter you want to change. A popup panel will ask you to enter a new value. After the refresh period of the OPC group to which the parameter is assigned, the value in the ReadBack column will change, indicating that the hardware has received it. If it is not the case, then either the connection to the Power Supply or the Power Supply itself is not working correctly. Just after opening the panel, if the Setting and ReadBack columns differ, it could be because somebody else is or was connected to the same channel and requested a different value. In this case you can ask to 'Load the settings from the hardware' into the Setting column. In the right table, the operational values read from the hardware are shown. With the Commands buttons (On/Off) it is possible to switch On and Off the Channel. The Action button allows to do the standard Framework Device operations: -Mask/Unmask/Acknowledge Alarms -Enable/Disable the channel
Restrictions	This panel works with standard CAEN channels. In addition to the standard channels, CAEN calls channels to other type of objects (e.g. power supply to Easy system boards). These other possibilities will be considered in a future version of the panel.



Dollar Parameters		
Name	**Description**	
$sDpNam	The name of the datapoint representing the CAEN channel.	Required

11. APPENDIX C – SUMMARY OF NAMING AND LOOK-AND-FEEL CONVENTIONS

This appendix has been moved to another document: *jcopFrameworkGuidelines – Shortlist.docx*