



# JCOP Framework

---

## Hierarchical Controls

### Configuration & Operation

Document Version: 1  
Document Date: 28-Jun-2001 (Updated 10-February-2004)  
Document Status: Draft  
Document Author: Clara Gaspar

---

#### Abstract

This document describes the integration of an FSM tool - SMI++ - with the JCOP Framework. The FwFSM Tools available allow the creation of hierarchies of Finite State Machines.

## 1 Introduction

The hierarchical controls part of the framework allows the definition and operation of hierarchies of objects behaving as Finite State Machines. This allows for the sequencing and automation of operations. In the following chapters we will describe briefly the architecture, the implementation and finally the tools available.

### 1.1 Controls Hierarchy Architecture

The mechanism adopted for modelling the structure of sub-detectors, sub-systems and hardware components in a consistent fashion is to use a hierarchical (tree like) structure. This tree is composed of two types of nodes: "Device Units" which are capable of monitoring and controlling the equipment to which they correspond and "Control Units" which are considered to contain Finite State Machine(s) which can model and control the sub-tree bellow them. As shown in Figure 1. In this Hierarchy "Commands" flow down and "Status and Alarm Information" flow up.

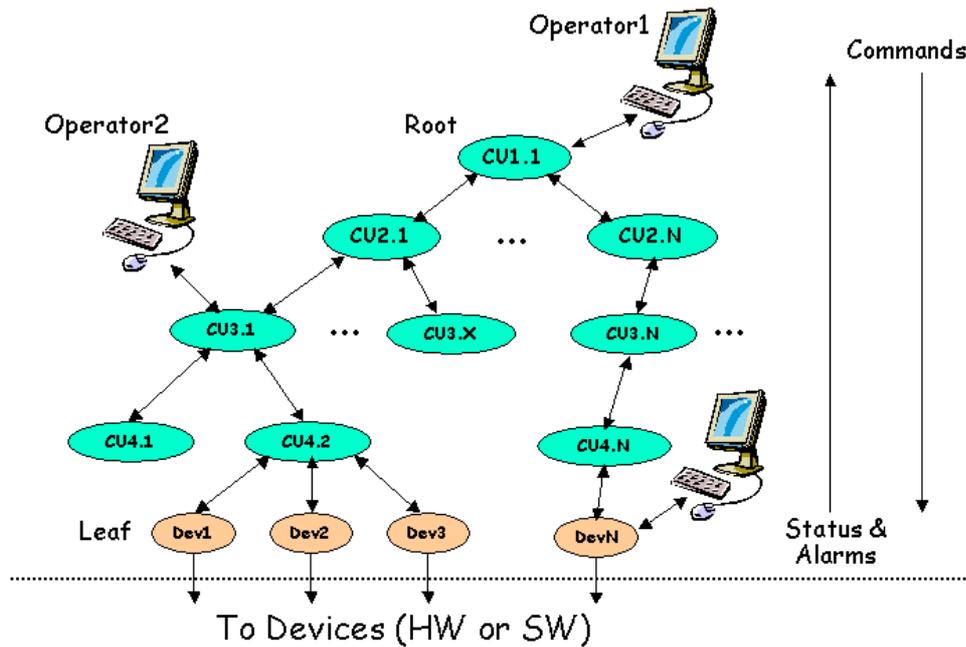


Figure 1: Generic Architecture

## 1.2 Components and their Interfaces

Each component in the tree (Device or Control Unit) provides Information and can receive Commands. From the point of view of hierarchical control, the Interface between Components and Components and Operators is "State" flowing up and "Command" flowing down - State/Command Interface.

## 1.3 Operators & Ownership

In order to be able to send commands to the different components an operator can reserve the whole tree or a sub-tree in which case he/she becomes the "owner". Each component has one and only one owner at any time. All components of a sub-tree have the same owner.

The components can receive commands from only one or from more operators depending on their Exclusivity mode:

- Exclusive mode - only the owner can send commands.
- Shared mode - any other operator with the correct access rights can also send commands.

Only the owner can change from one mode to the other.

## 1.4 Control Units

Control Units are logical decision units. They can take decisions and act on their children (i.e. send them "Commands") based on their "States". Any Control Unit and the associated sub-tree can be a self-contained entity. The logic behaviour of a Control Unit is expressed in terms of Finite State Machines. State transitions can be triggered by:

- Command Reception (either from its parent or from an operator)
- State changes of its children

State transitions cause the evaluation of logical conditions and possibly "Commands" to be sent to the children.

This mechanism can be used to propagate actions down the tree, to automate operations and to recover from error situations.

## 1.5 Device Units

Device Units implement the interface with the lower level components (Hardware or Software). They are always a tree "leaf" (i.e. they have no children). They do not implement logic behaviour. They receive:

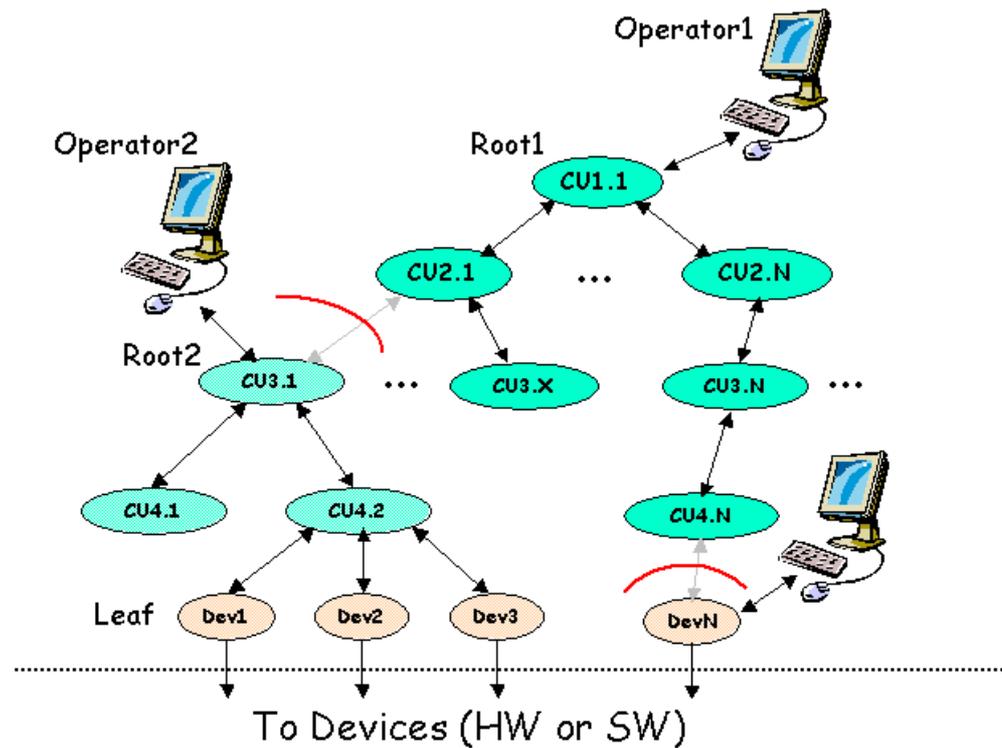
- "Commands" and act on the device
- device data and translate it into a "State".

## 1.6 Partitioning Modes

Each Control Unit knows how to partition "out " or "in" its children. Excluding a child from the hierarchy implies that it's state is not taken into account any more by the parent in its deciding process, that the parent will not send commands to it and that the owner operator releases ownership so that another operator can work with it. Only the owner can exclude a component from the hierarchy.

It was felt that Excluding completely a part of the tree was not enough so the following Partitioning Modes were defined, as in the graphical representation of Figure 3.:

- Included - A component is included in the Hierarchy, it receives "Commands" from and sends its "State" to its parent. It has the same owner as its parent.
- Excluded - A component is excluded from the hierarchy, it does not receive "Commands" and its "State" is not taken into account by its parent. It has no owner. The component is either faulty or ready to work in stand-alone, for calibration, tests, etc.



- StandAlone - A component is working in stand-alone, it does not belong to the hierarchy anymore (it became the root of a new hierarchy) and has a new owner.

Figure 2: Partitioning Components

- CommandsDisabled - A component is partially excluded from the hierarchy, it does not receive "Commands" but its "State" is still taken into account by its parent. An expert wants to work on it (to fix quickly a problem) since the experiment will not continue until it is fixed.
- Manual - A component is partially excluded from the hierarchy, the expert is working on it. The expert is the owner, he/she wants to send commands in an exclusive way
- Ignored - A component can be ignored 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 is only partially faulty) and the operator wants to proceed.

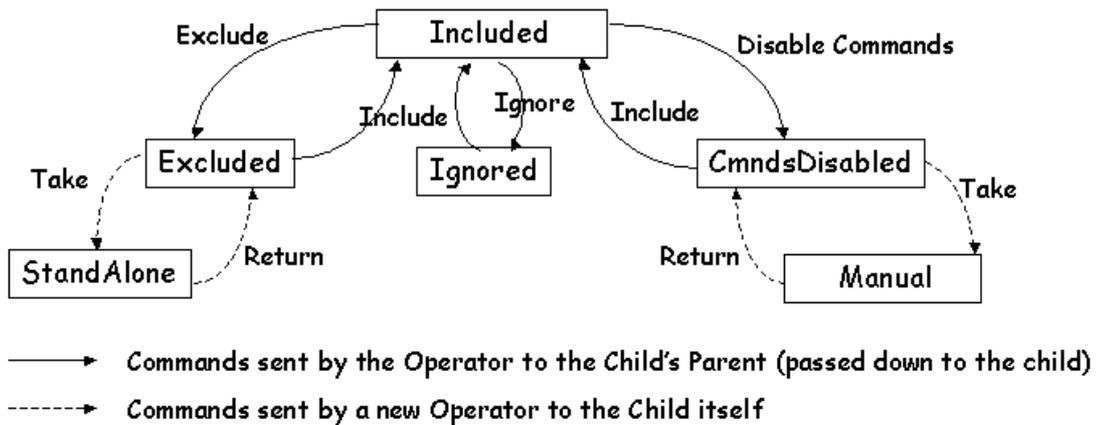


Figure 3: Partitioning Modes

## 2 Controls Hierarchy Implementation

The hierarchy is composed of Device Units and Control Units. In order to create and configure components of these types we need to know what they are:

**Device Units:** A Device Unit corresponds to a Datapoint of a certain Datapoint Type. Device Types containing an entry in `_FwDeviceDefinition` are automatically recognized as Framework Device Types.

**Control Unit:** Is a complex entity comprizing PVSS datapoints (containing information about label, panel, ownership, exclusivity modes, etc.) and Finite State Machine processes (providing information on objects, states, possible actions, etc.). PVSS communicates with the FSM processes via an API Manager - PVSS00smi.

The Finite State Machine (FSM) toolkit incorporated in the framework is called SMI++ (State Management Interface), very briefly, SMI allows the description of any sub-system as a collection of objects, each object behaving as a FSM, i.e., objects are always in a well-defined state and can receive actions that will make them transit to another state. A logically related group of objects (a sub-system) is called in SMI terms: a domain.

SMI defines two types of objects: abstract objects and physical objects:

- Abstract objects implement logic behaviour, they have a list of allowed states, in each state a list of allowed actions and when an action gets triggered (either by the reception of a command or by a state change of another object) they execute instructions like sending commands to other objects or testing the state of other objects. The behaviour of the object is coded using a very simple language called SML.
- Physical objects implement the interface to real devices, they also have a list of allowed states, and in each state a list of allowed actions, but when they receive a command they have to act on the device they model, and when the device's data changes they have to maybe change state. Physical objects can be coded in any language (C, C++, or using PVSS scripts).

A Device Unit corresponds to an SMI physical object.

A Control Unit corresponds to an SMI domain, i.e. it is composed of one or more (abstract and/or physical) objects.

SMI like PVSS allows the definition of object types and the derivation of objects from the type. So in order to create a hierarchy it is necessary to:

- Create any Device Types (i.e. the types from which Device Units will “inherit”)
- Create any Object Types (i.e. the types from which any abstract objects will “inherit”)
- Create Control Units (i.e. instantiate the devices and/or abstract objects and include them in a particular domain).

The following steps are necessary in order to create/configure a Device Type:

- Create a Datapoint Type (or use an existing one, the one that implements the device)
- Use the FSM Configuration Tool to:
  - Define which states this device can have
  - Define the allowed actions in each state
  - Define how the state is derived from the device's datapoint items
  - Define how the actions act on the device's datapoint items

The following steps are necessary in order to create/configure an Object Type:

- Use the FSM Configuration Tool to:
  - Create a new abstract object type
  - Define which states this object can have
  - Define the allowed actions in each state
  - provide the action “code” with the help of the “wizard”

The following steps are necessary in order to create/configure a Control Unit (i.e. an SMI domain)

- After having created the necessary device and object types
- Use the FSM Configuration Tool to:
  - Create a new domain (either as tree root or under another one)

- Add Objects and /or Devices to it by:
  - choosing the Object/Device type
  - providing a name

The system is then ready to be operated, the Control Unit domains can be started (or stopped, etc.) and generic panels to visualize and control them and their objects and devices are available.

### 3 The FSM Configuration & Operation Tool

#### 3.1 Defining Device Types and Object Types

The screenshot shows a software window titled "Vision\_1: fwFSM\FSMConfi...". It has a menu bar with "File", "Panel", and "?". Below the menu bar is a toolbar with icons for file operations and help. The main area is titled "FSM Configuration:" and contains three tabs: "Objects", "Domains", and "Operation".

Under the "Objects" tab, there are two sections:

- Device Types:** A list containing "DAQDevice" (highlighted) and "DCSDevice".
- Logical Object Types:** A list containing "DAQTreeNode" and "DCSTreeNode" (highlighted). To the right of this list is a checkbox labeled "Fw".

Below the lists is a text input field containing "DCSTreeNode". At the bottom of the configuration area are two buttons: "Create" and "Delete". At the very bottom of the window is a "Close" button.

Two callout boxes provide definitions:

- The top callout box points to the "Device Types" list and contains the text: "Device Types: Represent User Devices, they correspond to Data Point Types created by the User which contain a folder called 'fwDeclarations' of type reference '\_FwDeclarations'. This folder is the interface to the above levels of the Control System)."
- The bottom callout box points to the "Logical Object Types" list and contains the text: "Logical Object Types: Represent abstract objects which contain the logical behaviour of the system. They are dynamically created."

### 3.2 Configuring Device Types

The screenshot shows a configuration dialog for a DAQDevice. At the top, 'Device Type' is set to 'DAQDevice' and 'Panel' is 'DAQDevice.pnl'. A 'Simple Config' button is present. Below are two lists: 'State List' containing RUNNING, READY (highlighted), NOT\_READY, and ERROR; and 'Action List' containing START. Below these lists are input fields for 'State' (READY) and 'Color' (blue), and 'Action' and 'NV' (checkbox). 'Add' and 'Remove' buttons are provided for each list. At the bottom, a 'Configure Device' section has 'Configure Device States' and 'Configure Device Actions' buttons, along with 'OK' and 'Cancel' buttons.

List of states the Device can be in, these can be added or removed directly in this panel or by clicking on "Simple Config".

The panel specific to this device type (will be called at run-time). By default the panel has the same name as the device type, but it can be changed here.

List of Actions that the Device can accept in each state, these can be added or removed directly in this panel or by clicking on "Simple Config".

Each state has a color to be seen at run-time

After having defined the states and actions of the object, click here to specify how the state is derived from the Device's DataPoint items and how the actions act in the Device's DataPoint items (if not done by the "Simple Config").

### 3.3 Simple Device Config

The "Simple Config" allows the definition of the States (and state colors) of the Device, of the actions allowed in each state and for each action the expected end-state.

**Device Type Configuration:**

State	Color	Actions	Transitions
RUNNING	Green	STOP (Remove) READY (Add)	STOP -> READY FAKE_ERROR -> ERROR
READY	Blue	START (Remove) RUNNING (Add)	START -> RUNNING
NOT_READY	Yellow	CONFIGURE (Remove) READY (Add)	CONFIGURE -> READY
	Orange		
ERROR	Red	RECOVER (Remove) NOT_READY (Add)	

State depends on DP Item: status

Actions act on DP Item: status

Specify here what Datapoint item is used to compute the new state and what datapoint item is set when an action arrives. This panel will generate simple scripts, for more complex behaviour please use the "Configure Device" button on the previous panel.

Apply Cancel

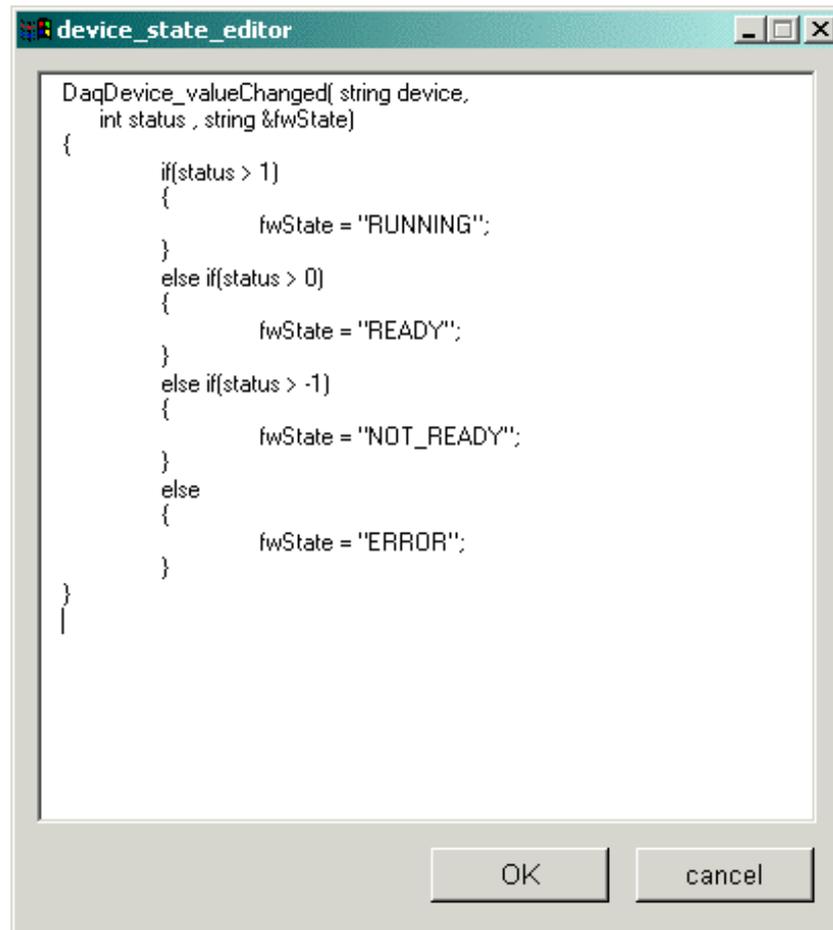
### 3.4 Complex Device Config: from DP Items to Device States

The screenshot shows a window titled 'config\_device\_states' with a configuration interface. It features four conditional rows:

- if** status [dropdown] float > 1 **then** RUNNING (indicated by a green vertical bar)
- else if** status [dropdown] float > 0 **then** READY (indicated by a blue vertical bar)
- else if** status [dropdown] float > -1 **then** NOT\_READY (indicated by a yellow vertical bar)
- else** ERROR (indicated by a red vertical bar)

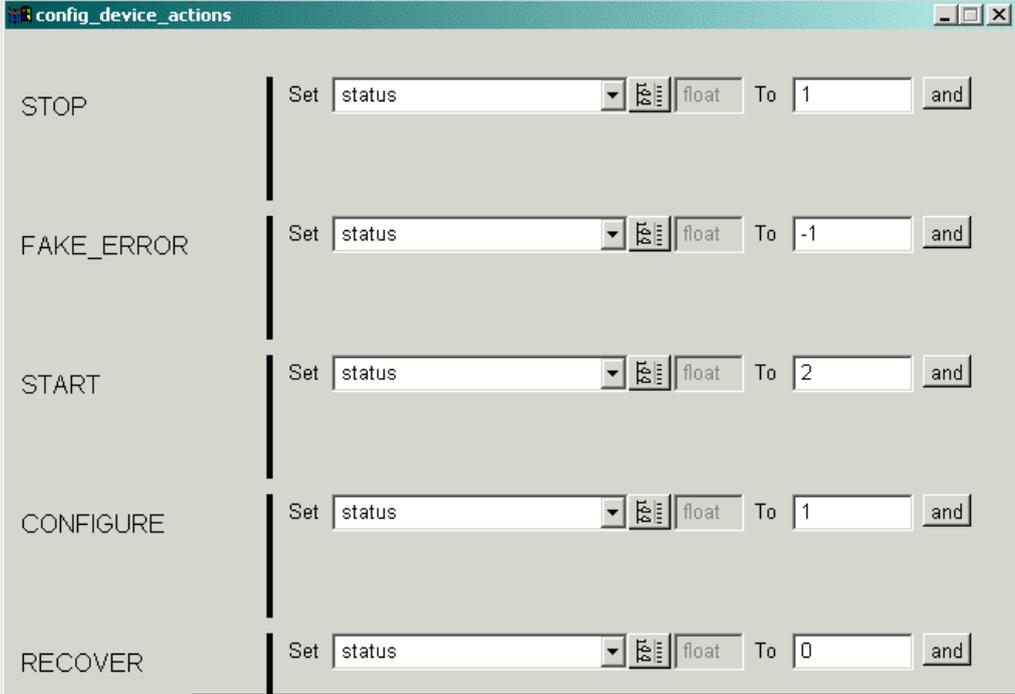
Each row includes a 'DP' icon and a 'then' dropdown menu. A text box at the bottom provides instructions: 'Configuring device states: Select the item(s) and what values it/they should have in order for the device to be in a particular state. Clicking on "Apply" will generate a script that you can modify by clicking on the "Complex Parametrization" button. Note: clicking on "Apply" after having modified the script will overwrite it.' Below the text box are three buttons: 'Apply', 'Complex Parametrization', and 'Close'.

### 3.5 An example Script that calculates the new State



```
DaqDevice_valueChanged( string device,
                        int status , string &fwState)
{
    if(status > 1)
    {
        fwState = "RUNNING";
    }
    else if(status > 0)
    {
        fwState = "READY";
    }
    else if(status > -1)
    {
        fwState = "NOT_READY";
    }
    else
    {
        fwState = "ERROR";
    }
}
```

### 3.6 Complex Device Config: from Device Actions to DP Items



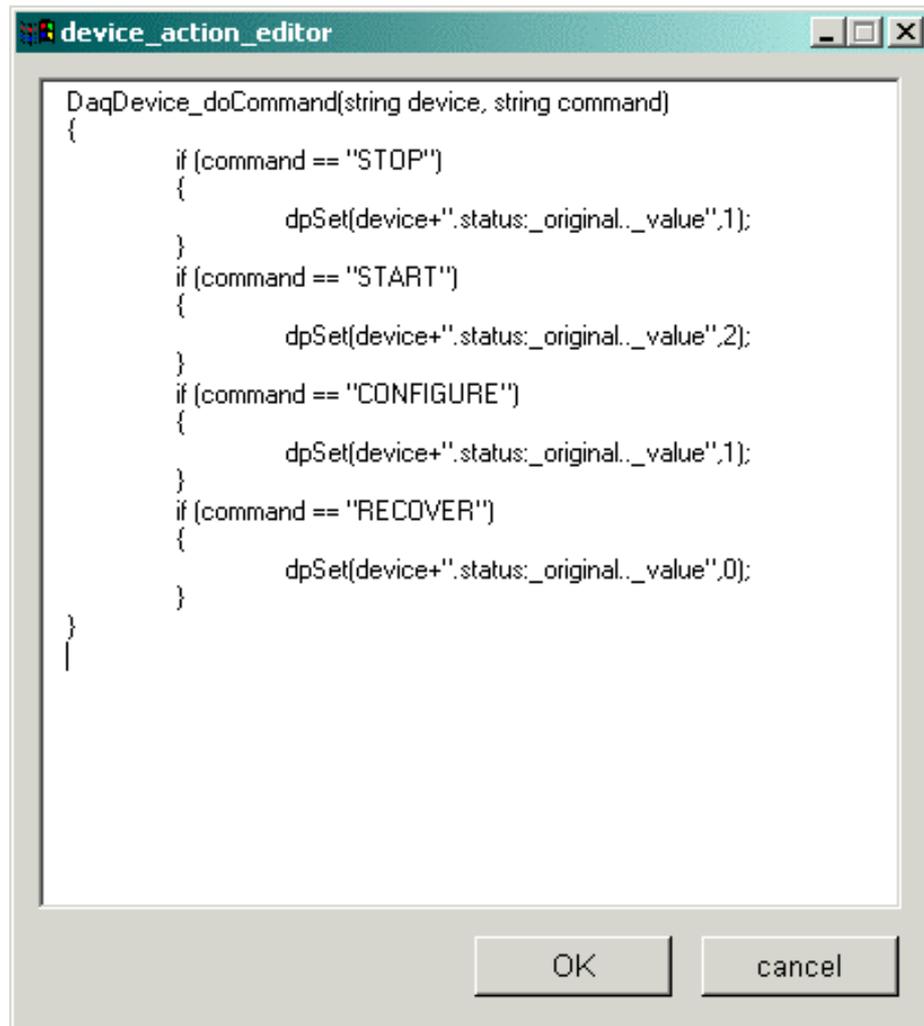
The screenshot shows a window titled "config\_device\_actions" with a list of actions on the left and their corresponding configuration on the right. Each action has a "Set" field with a dropdown menu (currently showing "status"), a "float" type indicator, and a "To" field with a numerical value. An "and" button is next to each "To" field.

Action	Set	Type	To
STOP	status	float	1
FAKE_ERROR	status	float	-1
START	status	float	2
CONFIGURE	status	float	1
RECOVER	status	float	0

Configuring device actions: Select the item(s) and what values it/they should be set to when an action is received. Clicking on "Apply" will generate a script that you can modify by clicking on the "Complex Parametrization" button.  
Note: clicking on "Apply" after having modified the script will overwrite it.

Buttons: Apply, Complex Parametrization, Close

### 3.7 An example Script called when an Action is received



```
DaqDevice_doCommand(string device, string command)
{
    if (command == "STOP")
    {
        dpSet(device+".status:_original._value",1);
    }
    if (command == "START")
    {
        dpSet(device+".status:_original._value",2);
    }
    if (command == "CONFIGURE")
    {
        dpSet(device+".status:_original._value",1);
    }
    if (command == "RECOVER")
    {
        dpSet(device+".status:_original._value",0);
    }
}
```

The image shows a dialog box titled "device\_action\_editor" with a standard Windows-style title bar (minimize, maximize, close buttons). The main area contains a C++ code snippet. The code defines a function named "DaqDevice\_doCommand" that takes two parameters: "string device" and "string command". The function body consists of four conditional blocks, each corresponding to a specific command: "STOP", "START", "CONFIGURE", and "RECOVER". Each block calls the "dpSet" function with a device-specific path and a numerical value. The "STOP" block sets the value to 1, "START" to 2, "CONFIGURE" to 1, and "RECOVER" to 0. The function is enclosed in curly braces. At the bottom of the dialog box, there are two buttons: "OK" and "cancel".

### 3.8 Configuring Object Types

The screenshot shows a configuration window for an object type named **DCSTreeNode**. The window includes a **Simple Config** button, a **Copy from type:** dropdown menu set to **DemoNode**, and several lists for configuration. Callouts provide detailed instructions for each section.

**State List:** NOT\_READY, READY, ERROR. Callout: "List of states the Object can be in, these can be added or removed directly in this panel or by clicking on 'Simple Config'".

**Action List:** CONFIGURE. Callout: "List of Actions that the Object can accept in each state, can also be done in 'Simple Config'. By double clicking on each action you can see/modify the FSM code corresponding to this action".

**When List:** when ( \$ANY\$FwCHILDREN in\_state ERROR ) do NV\_GOTO\_EI; when ( \$ALL\$FwCHILDREN in\_state READY ) do NV\_GOTO\_RE. Callout: "List of conditions that can trigger an action or a state change while in this state. By clicking on 'Add' you can specify a new 'when' condition, by double clicking on one of them you can modify it. 'Simple Config' can also generate conditions."

**State/Action/Condition Editor:** State: NOT\_READY, Color: [Blue], Action: [Empty], NV: [Empty]. Callout: "Each state has a color to be seen at run-time".

**Panel:** DCSTreeNode.pnl. Callout: "The panel specific to this object type (will be called at run-time). By default the panel has the same name as the object type, but it can be changed here."

Buttons: Add, Remove, OK, Cancel.

### 3.9 Simple Object Configuration

The "Simple Config" allows the definition of the States (and state colors) of the Device, of the actions allowed in each state and for each action the expected end-state.

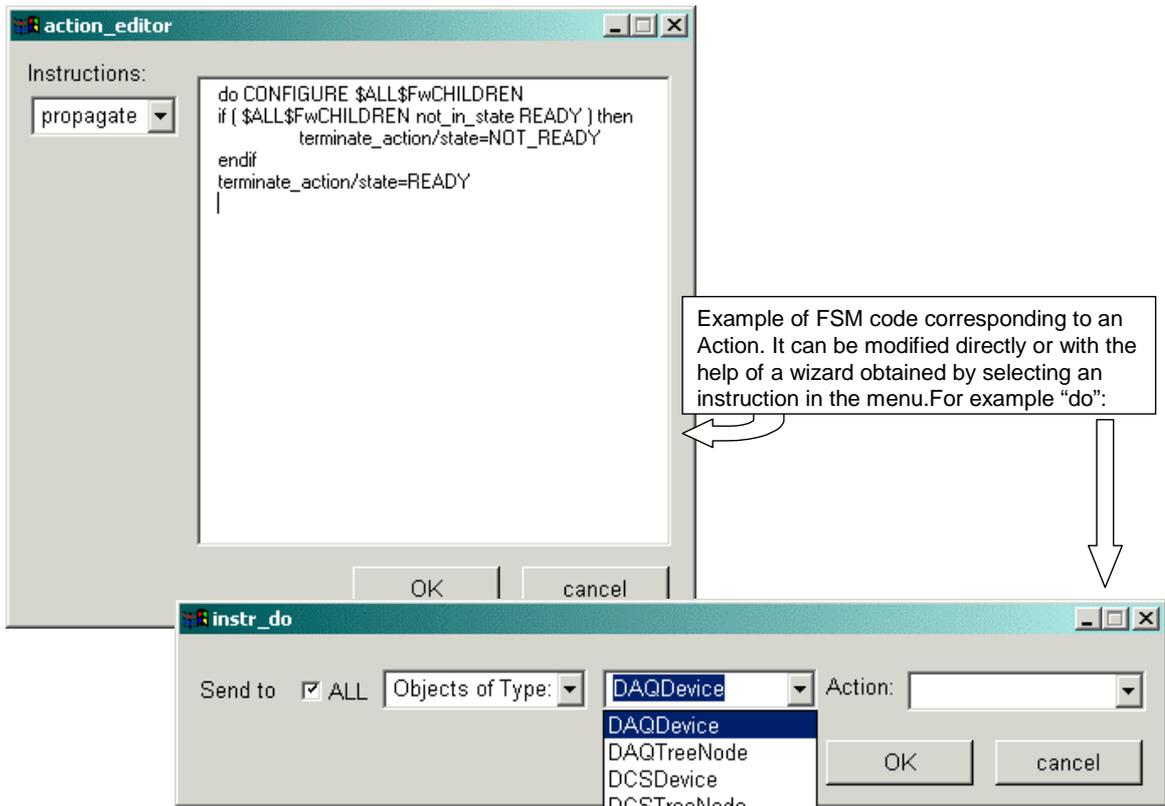
Initial State	Color	Actions
<input checked="" type="radio"/> READY	Green	RESET → NOT_READY
<input type="radio"/> NOT_READY	Blue	CONFIGURE → READY
<input type="radio"/>	Yellow	
<input type="radio"/>	Orange	
<input type="radio"/> ERROR	Red	RECOVER → NOT_READY

This panel will generate the FSM code for the object, you can view it and modify it for more complex behaviour by double clicking on each state in the previous panel.

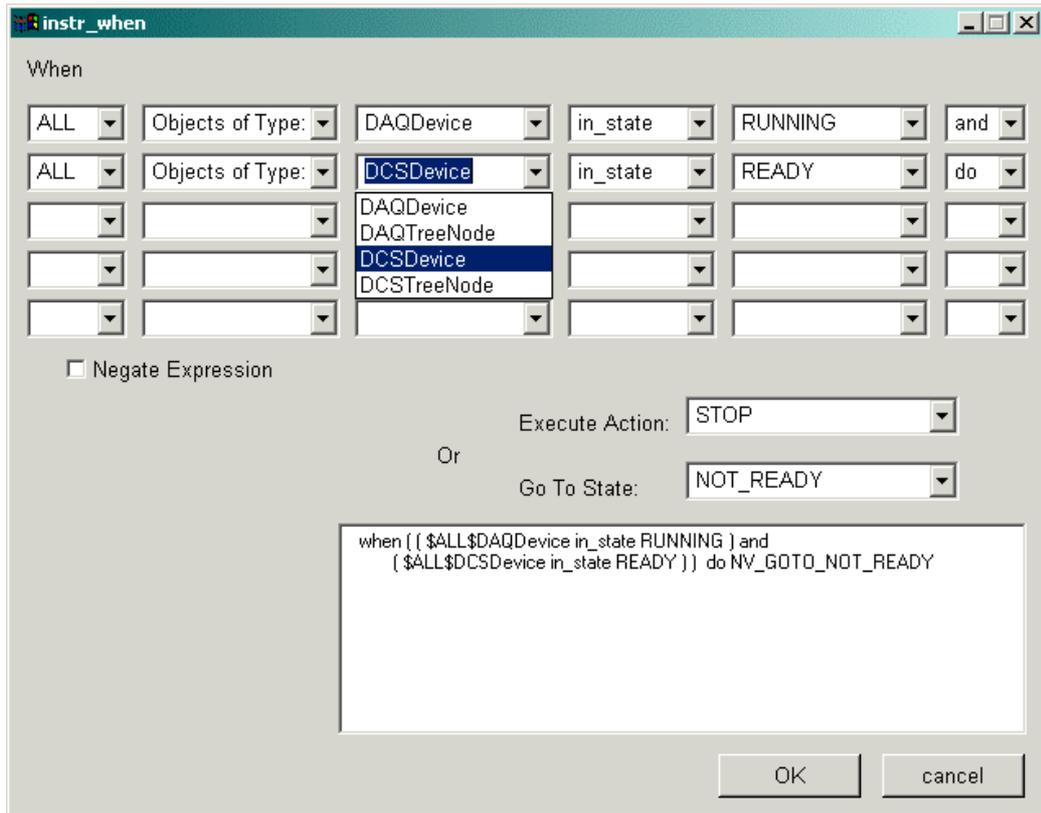
Apply Cancel



### 3.10 Example FSM Action code



### 3.11 Example FSM “when” condition wizard



### 3.12 Configuring Hierarchies of FSM Domains

This tree represents a hierarchy of FSM domains. Each domain can be expanded or collapsed by double clicking on it. You can Add/Remove domains using the “Add” and “Remove” buttons or the right mouse button when the domain is selected.

By double clicking here (on the domain name) you can configure it. The “CU” flag stands for “Control Unit” meaning this is a domain or just a device.

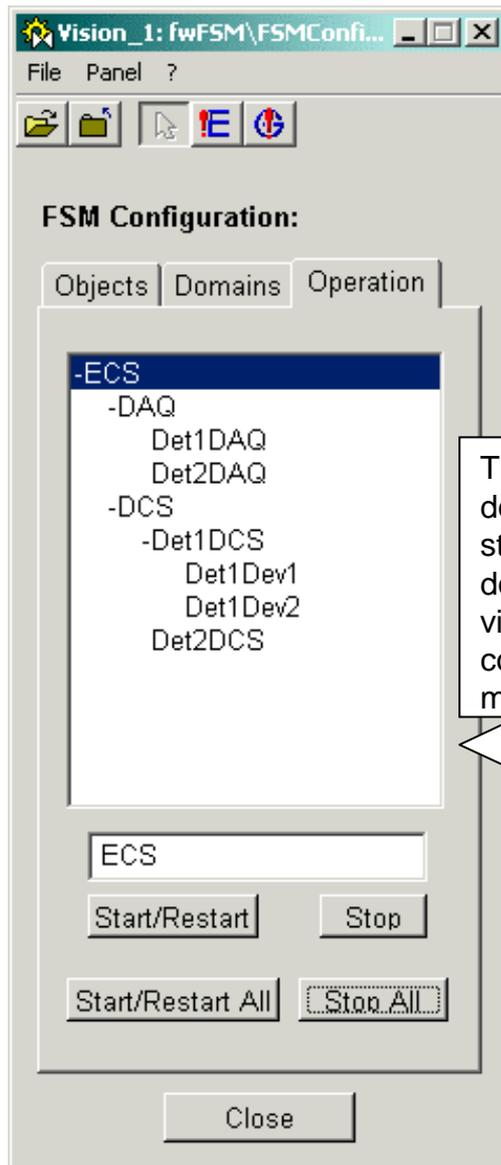
The domains have to be regenerated if the Object or Device types are modified.

### 3.13 Configuring FSM Domains

The screenshot shows a dialog box titled "smi\_domain" with the following fields and callouts:

- Domain:** Det1DCS
- Label:** Det1DCS (Callout: Label and Panel of this domain, to be used at run-time.)
- Panel:** DCSTreeNode.pnl
- Object List:** A list containing Det1DCS, Det1Dev1 (highlighted), and Det1Dev2 (Callout: List of Objects and Devices in this domain, to be added or removed by using the "Add" and "Remove" buttons)
- Object Type:** DCSDevice (dropdown)
- Object Name:** Det1Dev1 (dropdown)
- CU:**  (Callout: If an Object or Device is added with "CU" flag set, this means create also an independent domain for it)
- Buttons:** Add, Remove, Apply, Close

### 3.14 Operating the Hierarchy of Domains



This tree represents a hierarchy of FSM domains. Each domain can be started, stopped or restarted by itself or all domains at the same time. You can visualize the domains and send commands to them by using the right mouse button (once they are running).

### 3.15 Operating the FSM Domains

Domain's Children (other Domains, Objects or Devices) :  
Name, State and Partitioning mode  
Double Click on Name to open panel  
Click on State for sending actions  
Click on partitioning mode for changing mode

Domain's top level Object :  
Name, State and Partitioning mode  
Click on State for sending actions  
Click on partitioning mode for changing mode

User Specific Logo

The screenshot shows a software interface for managing the ECS system. At the top, the title bar reads 'ECS: System1: Manager3'. The main area is divided into several sections: a top header with 'System' (ECS) and 'State' (PHYSICS); a 'Sub-System' table; a 'Fill Number' and 'Run Number' display; a 'Trigger Rate' gauge; a 'Live Time' progress bar; and a 'Messages' log at the bottom. A 'Close' button is in the bottom right corner. Dashed boxes and arrows highlight specific UI elements and their functions as described in the text.

Sub-System	State
DAQ	RUNNING
DCS	READY
LHC	PHYSICS

Fill Number: 103  
Run Number: 234522  
Trigger Rate: 40 Hz  
Live Time: 100%

Messages  
11-Jun-2001 16:48:46 - Run 234522 Started

11/06/2001 16:50:16

Close

Domain Specific panel