European Laboratory for Particle Physics
Laboratoire Européen pour la Physique des Particules
CH-1211 Genève 23 - Suisse

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.

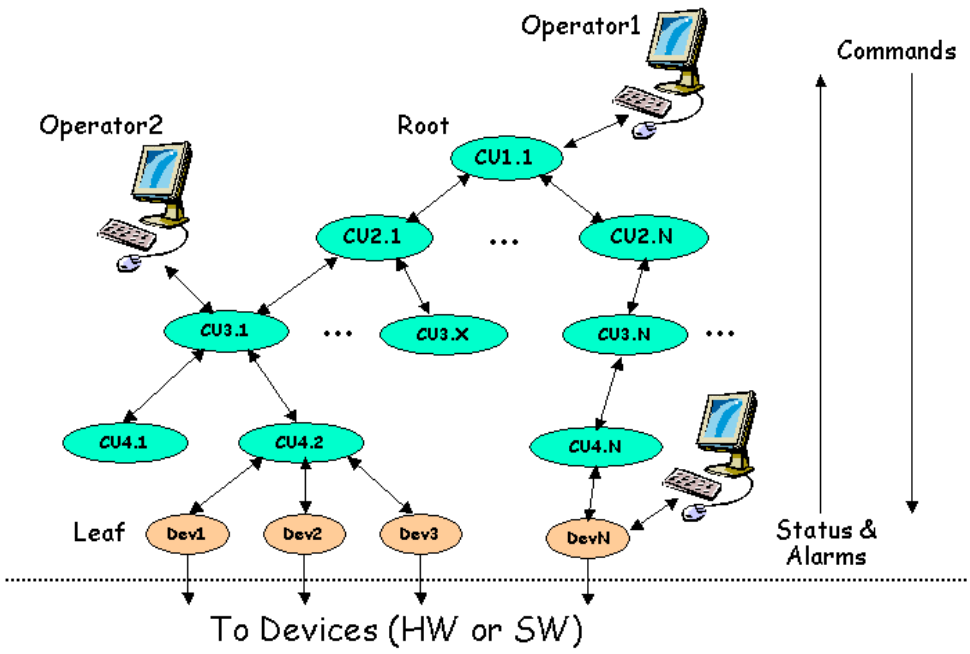


Figure 1: Generic Architecture

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:

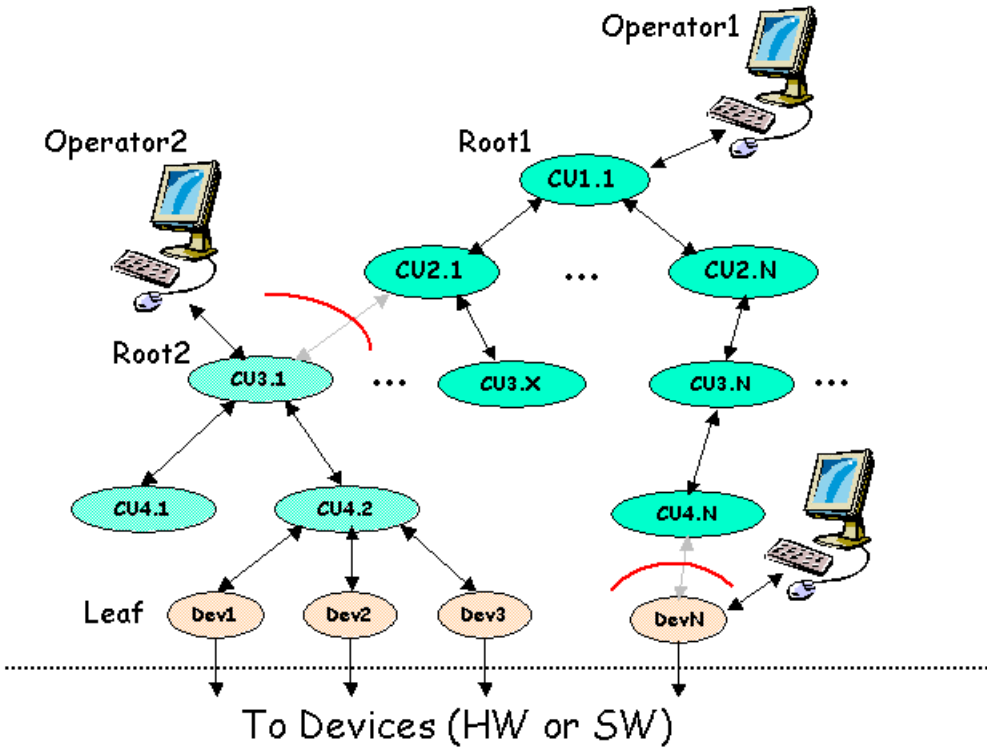
- "Commands" and act on the device
- device data and translate it into a "State".

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.

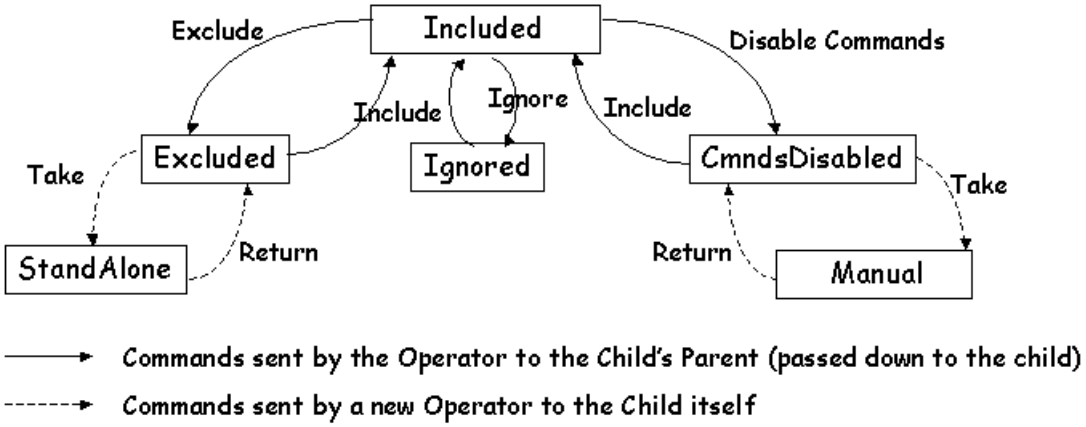


Figure 3: Partitioning Modes

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 allways 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 bahaviour, 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 af 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 abtract objects will "inherit")
- Create Control Units (i.e. instanciate 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 wich 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 wich 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)

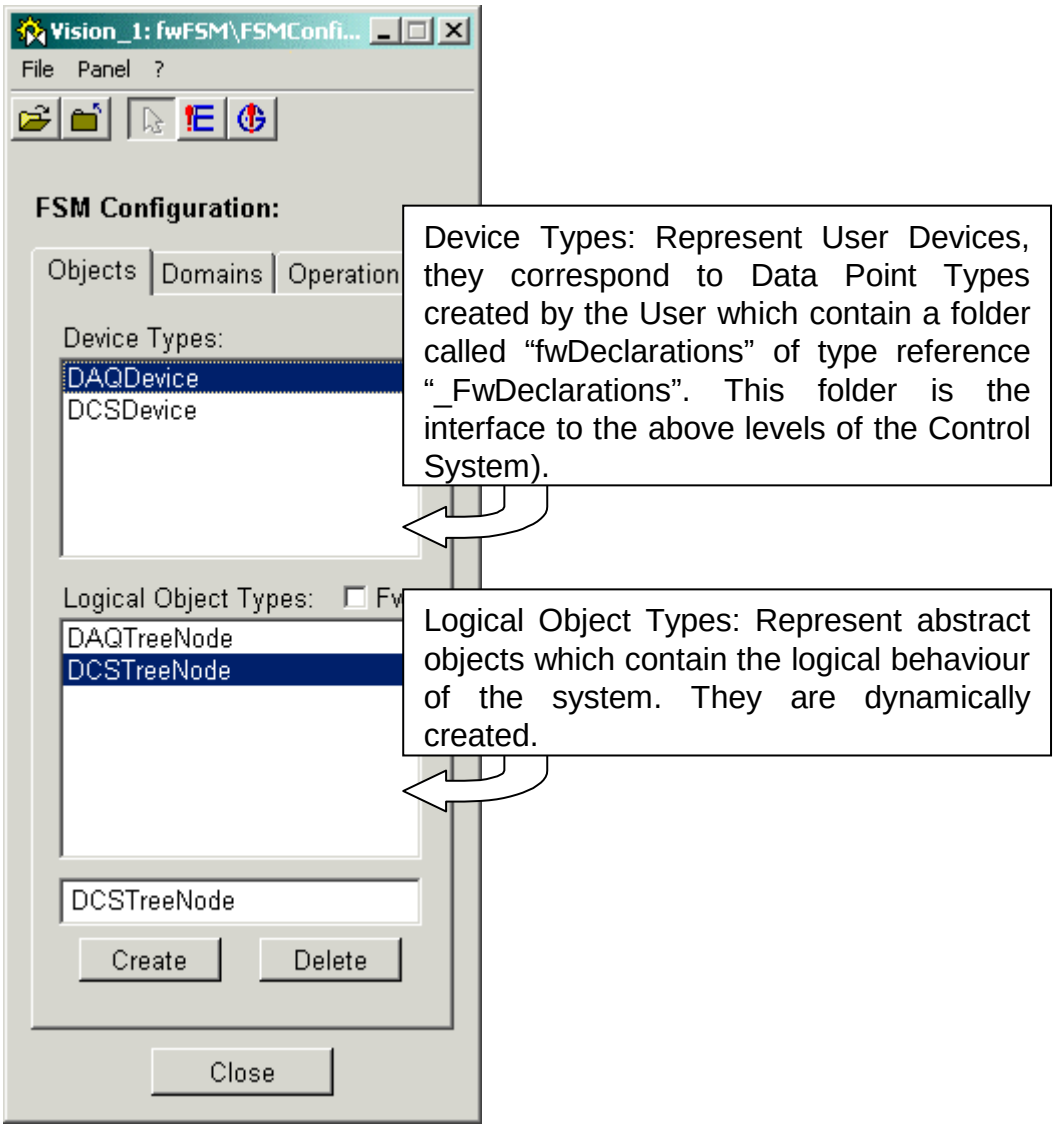
- After having created the necessary device and object types
- Use the FSM Configuration Tool to:
    - Create a new domain (either as tree root or under another one)

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

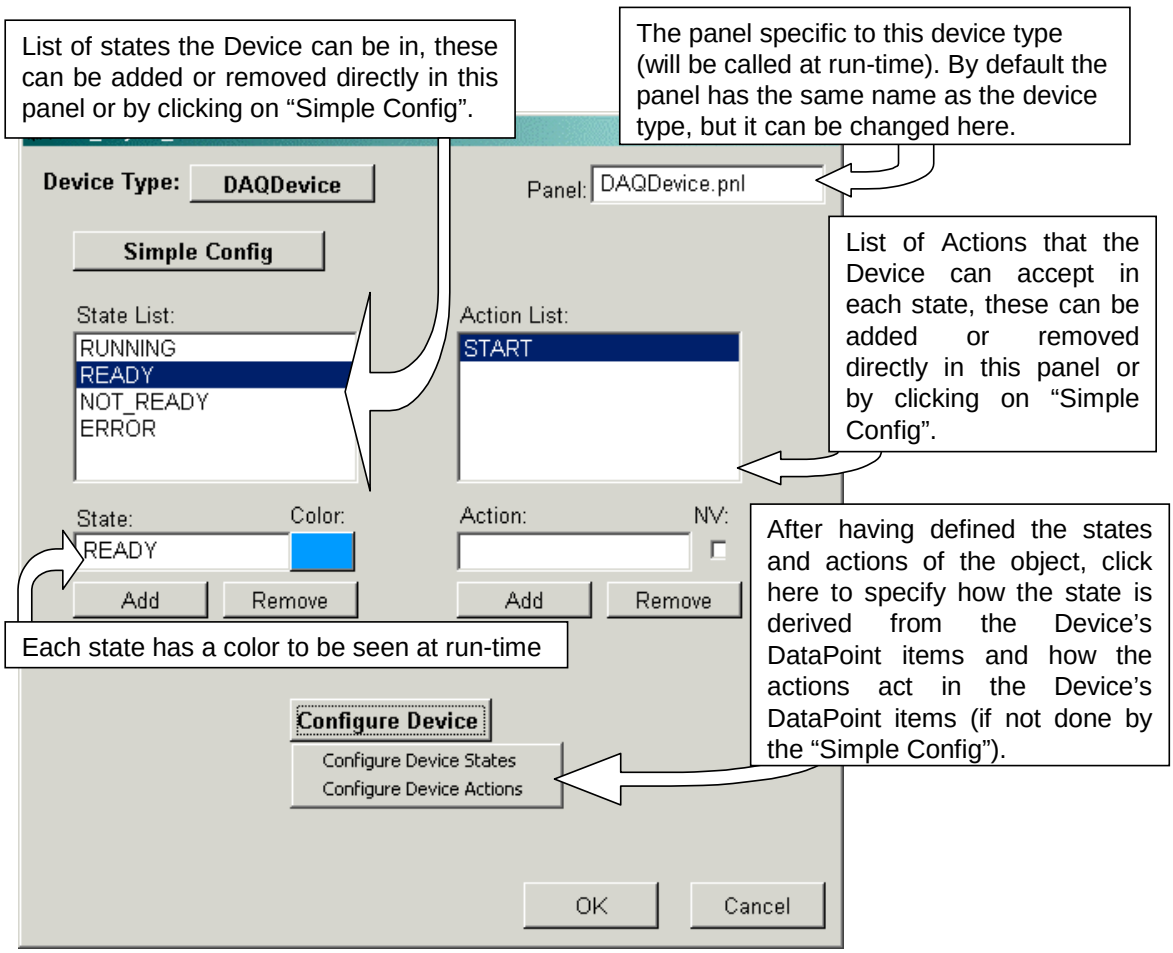
The system is then ready to be operated, the Control Unit domains can be started (or stopped, etc.) and generic panels to visualize and control them and their objects and devices are available.

3 The FSM Configuration & Operation Tool

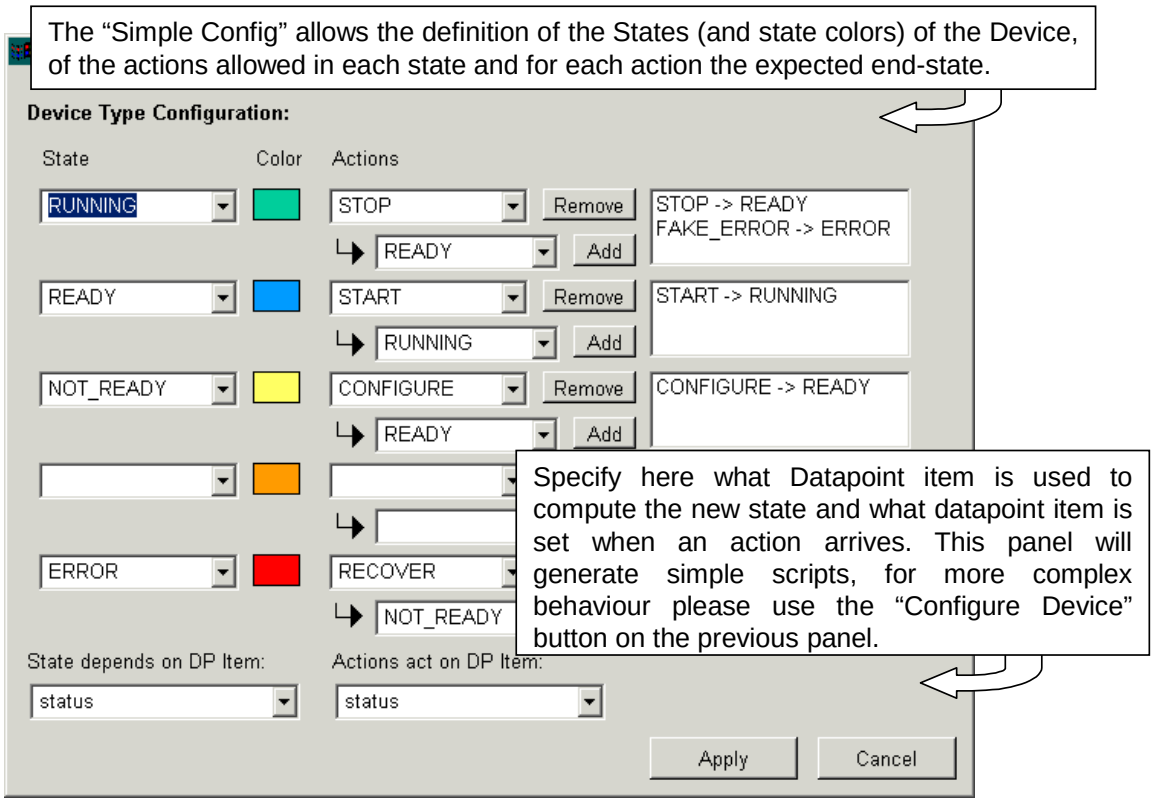
3.1 Defining Device Types and Object Types



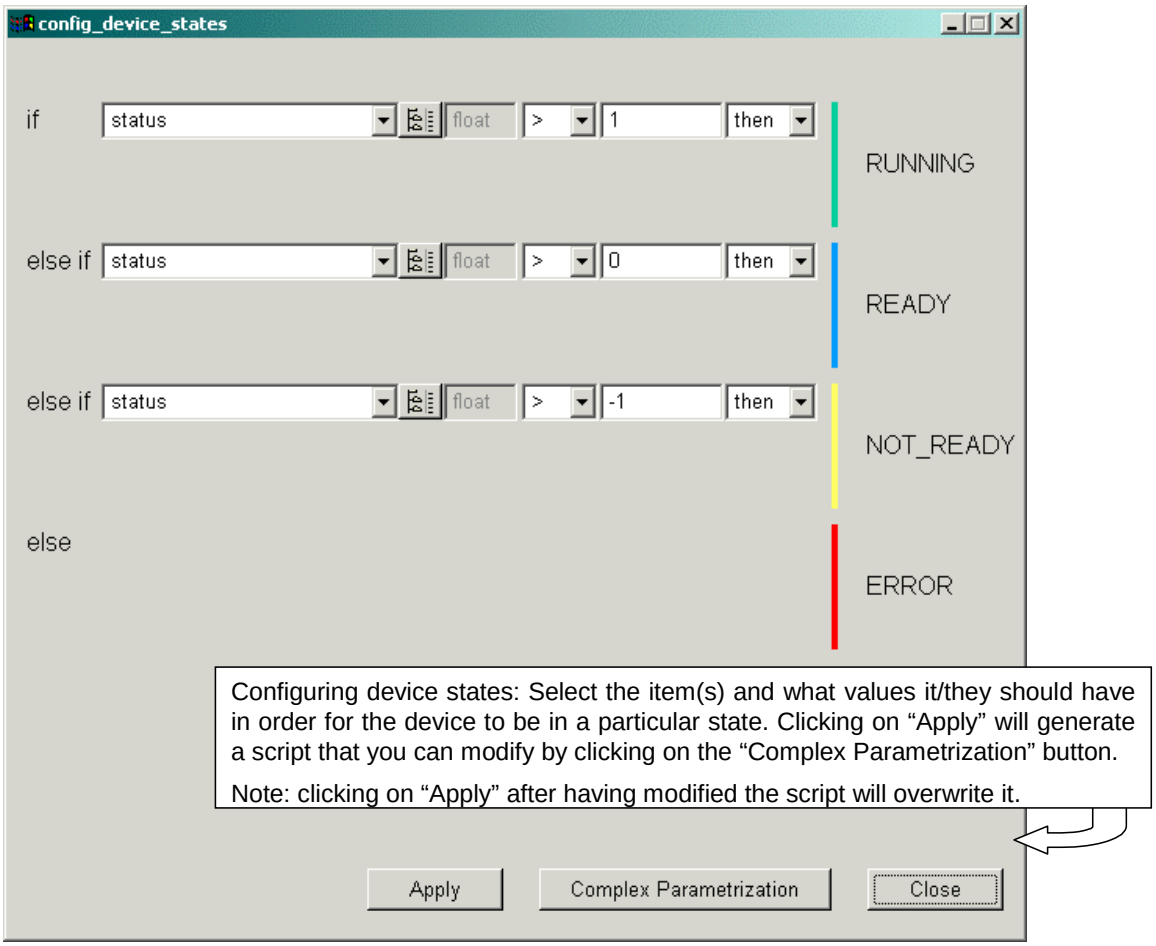
3.2 Configuring Device Types



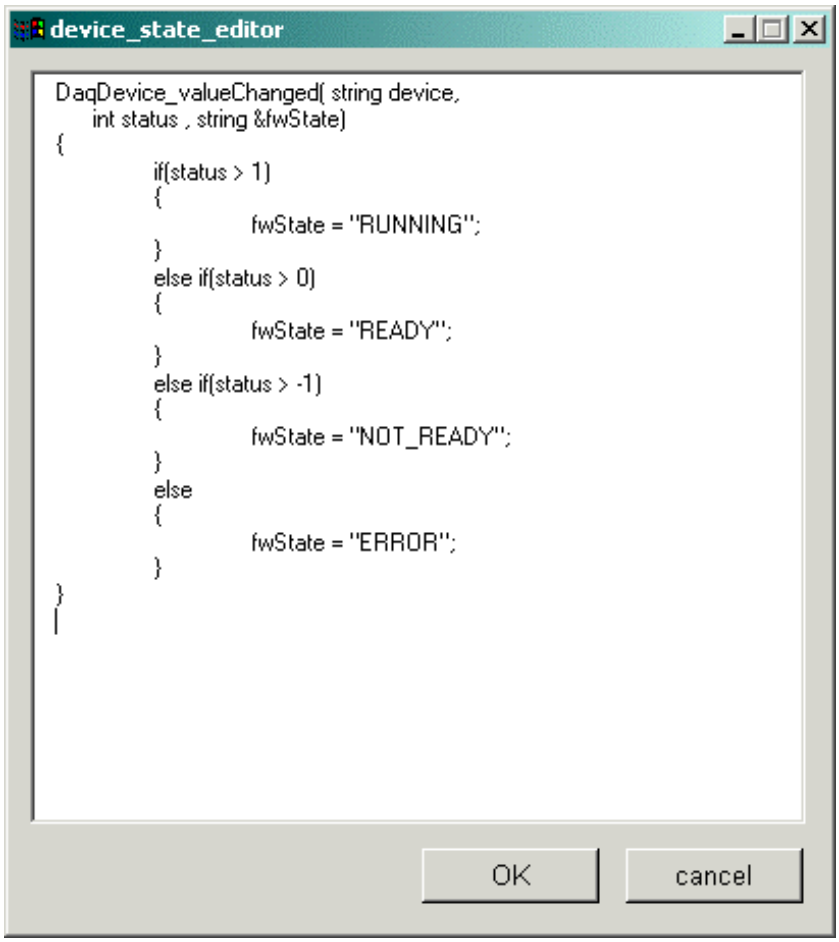
3.3 Simple Device Config



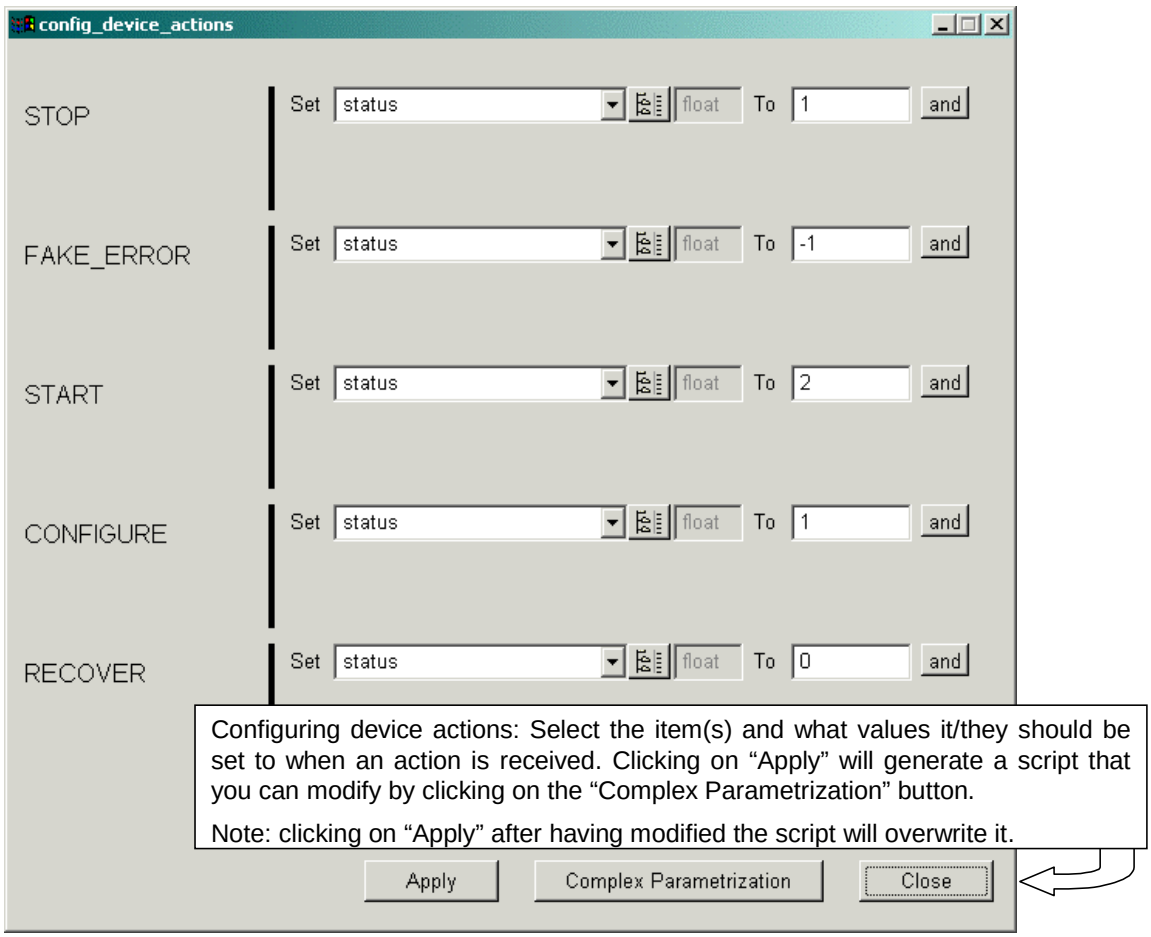
3.4 Complex Device Config: from DP Items to Device States



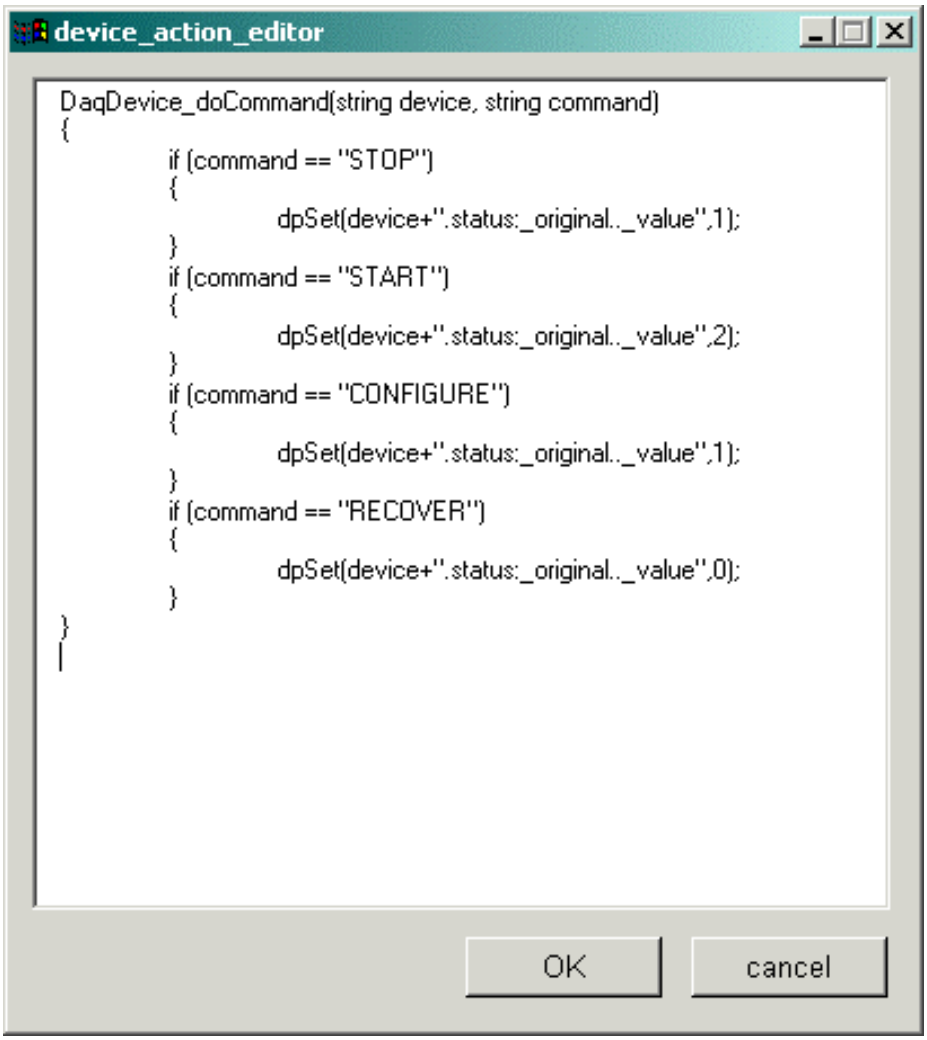
3.5 An example Script that calculates the new State



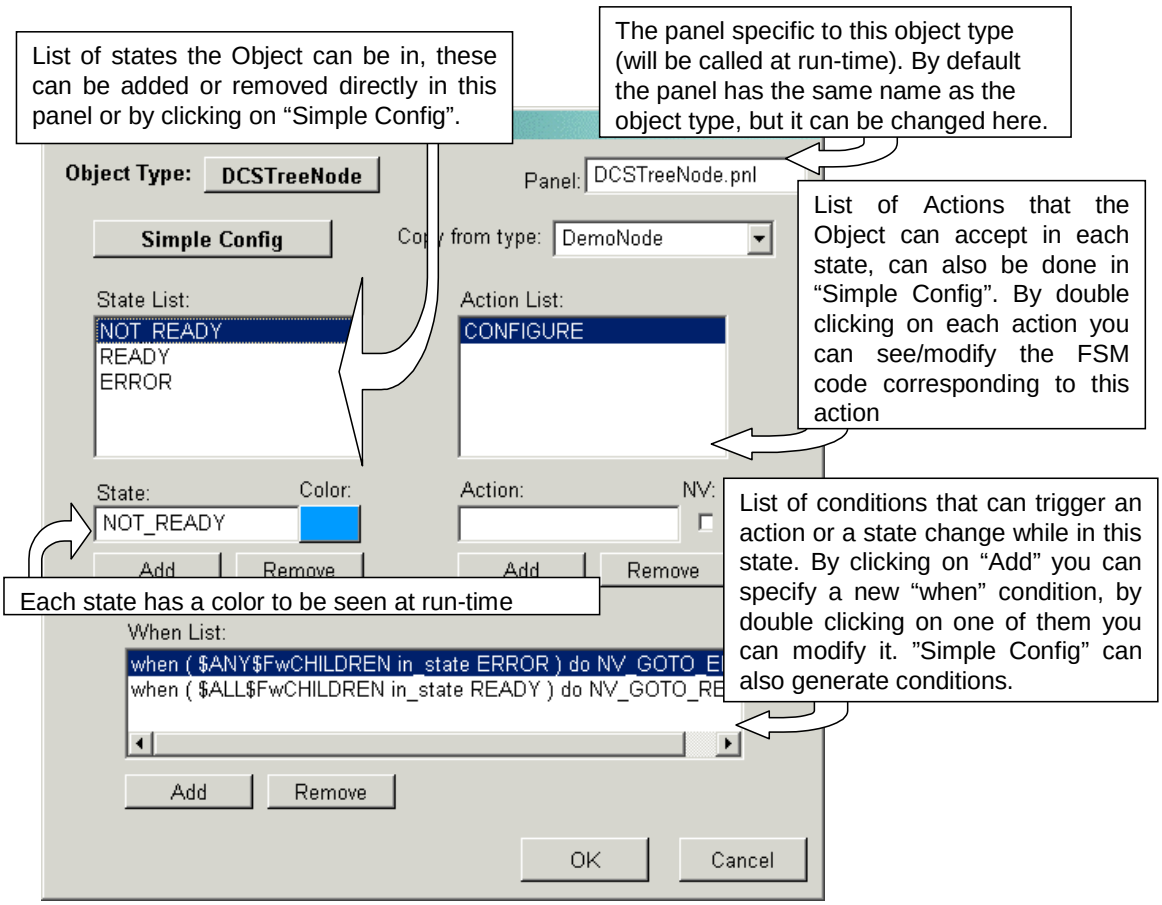
3.6 Complex Device Config: from Device Actions to DP Items



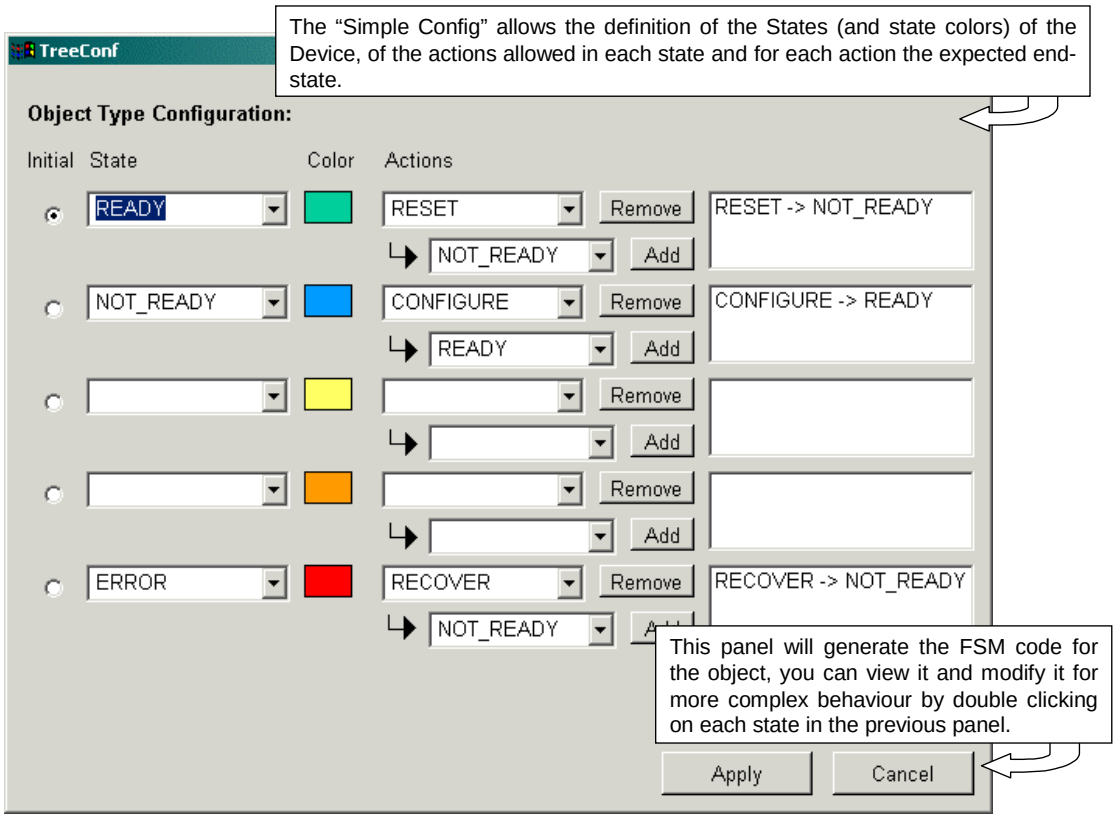
3.7 An example Script called when an Action is received



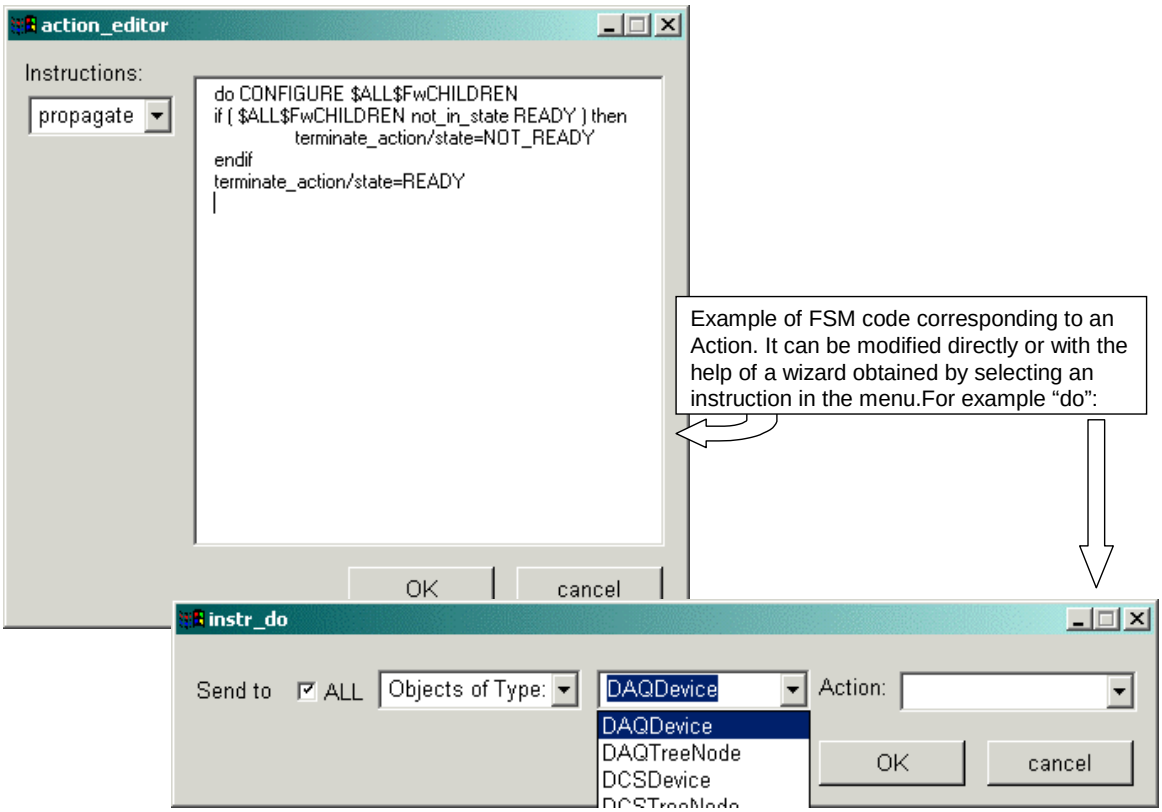
3.8 Configuring Object Types



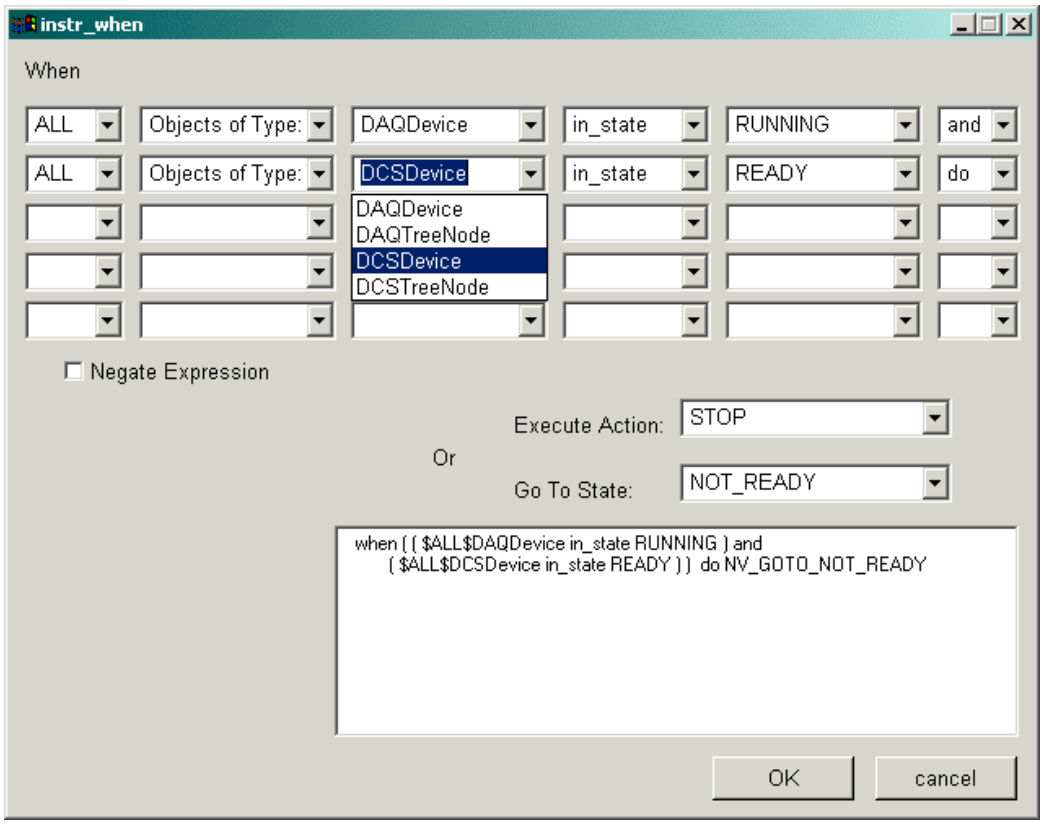
3.9 Simple Object Configuration



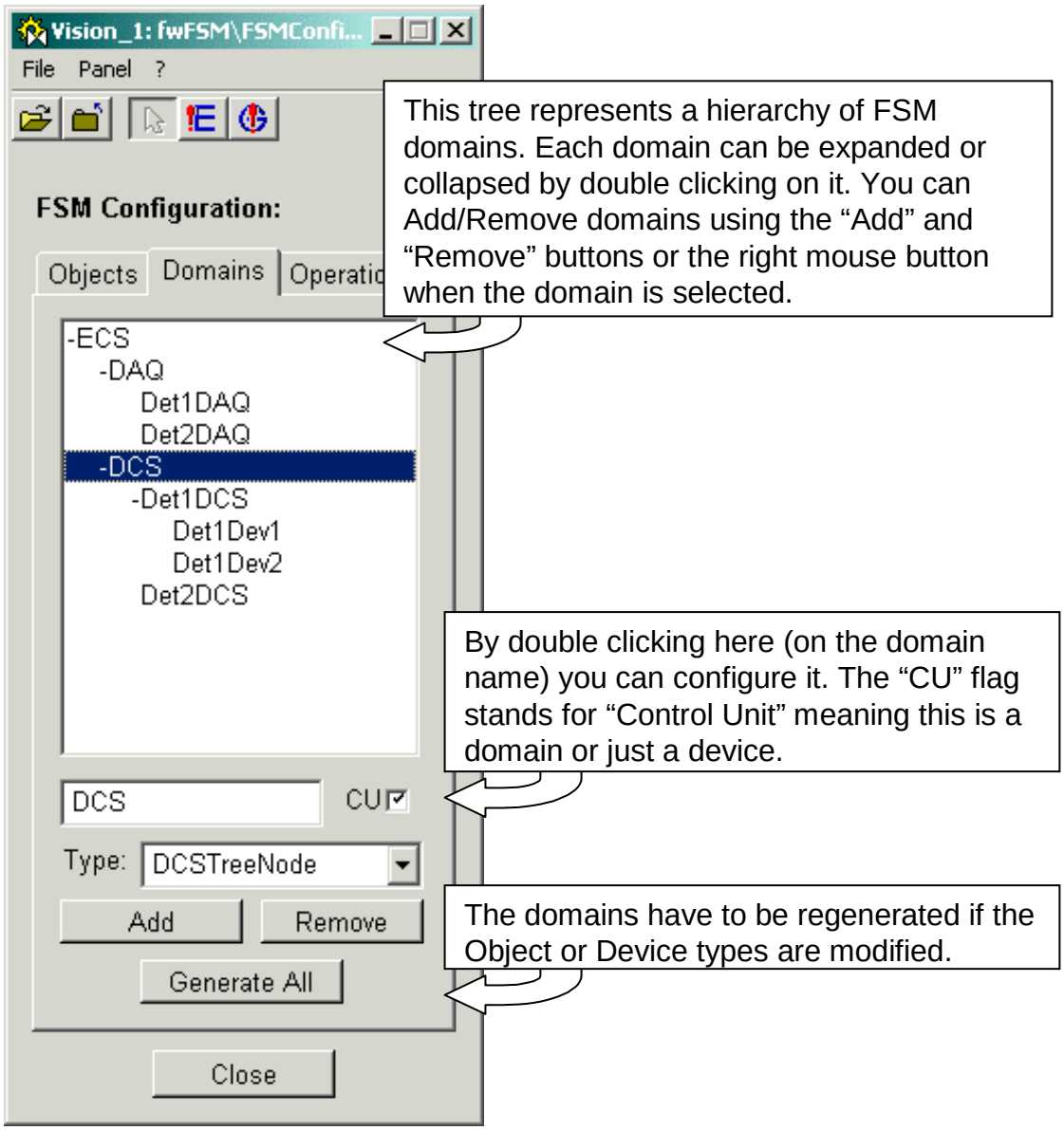
3.10 Example FSM Action code



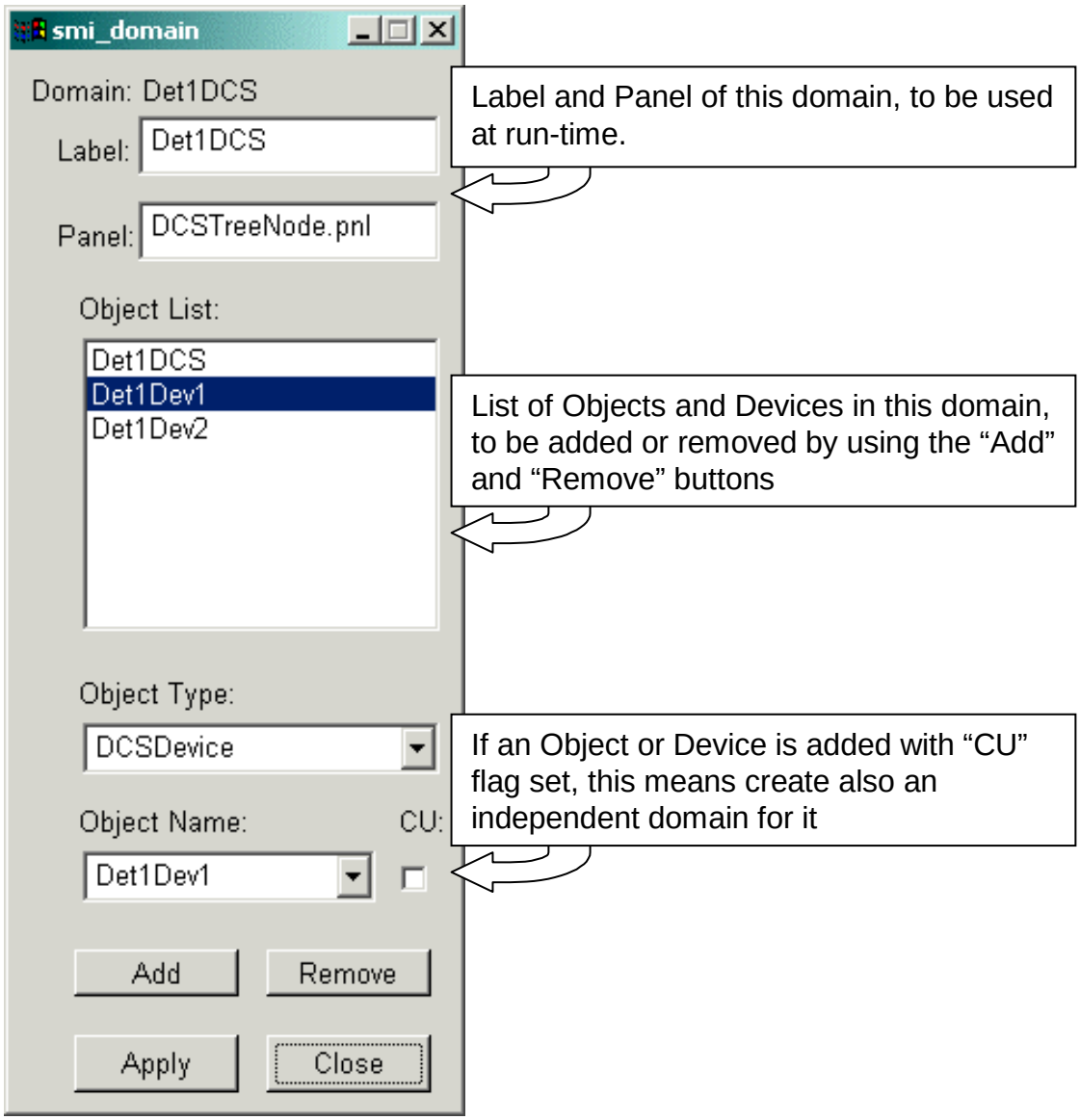
3.11 Example FSM “when” condition wizard



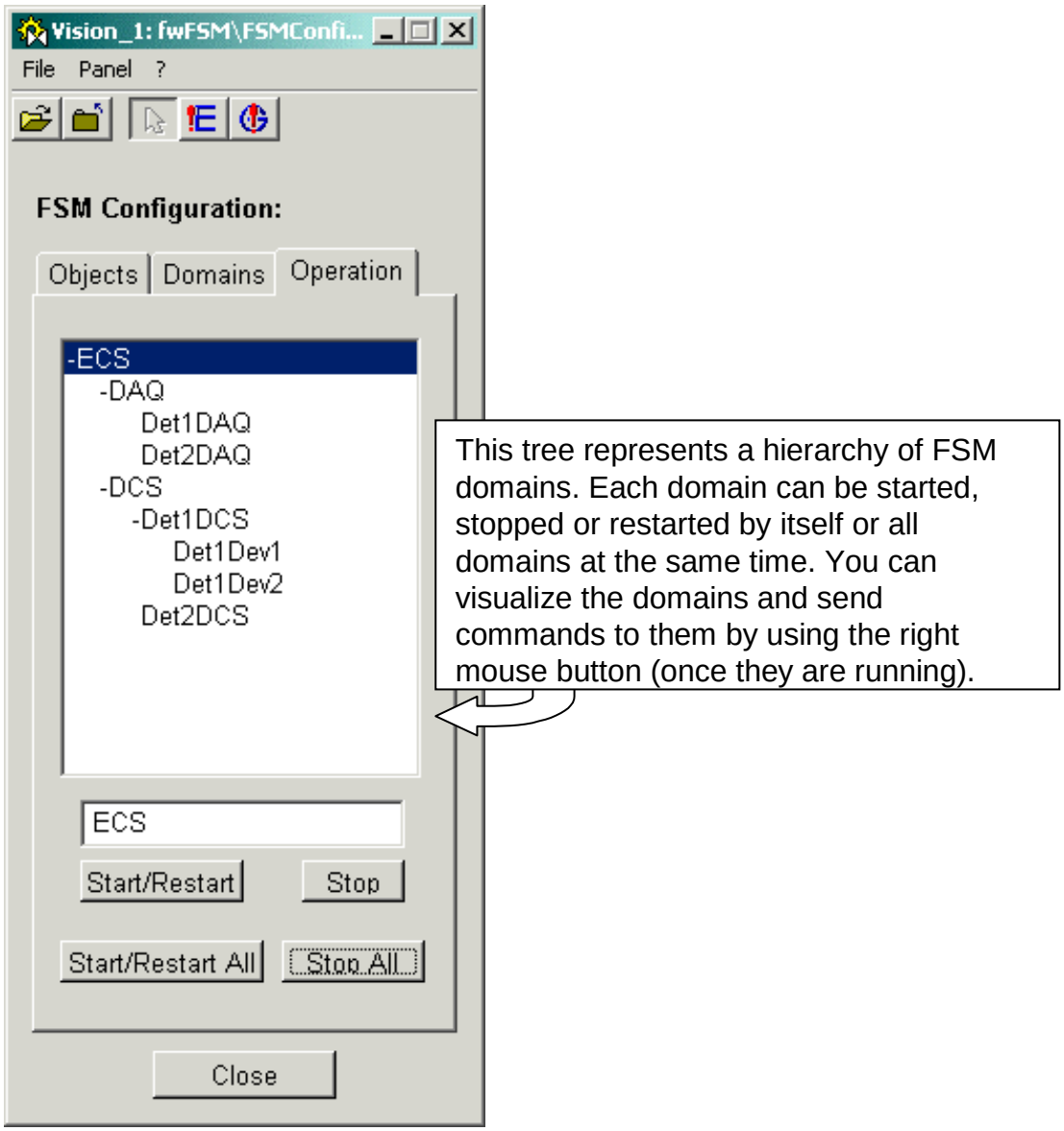
3.12 Configuring Hierarchies of FSM Domains



3.13 Configuring FSM Domains



3.14 Operating the Hierarchy of Domains



3.15 Operating the FSM Domains