# Class granularity

Added by Stephane Deghaye, last edited by Stephane Deghaye on Jun 26, 2008

**Under construction**
The info given here is under edition and reflects (ASAP) the latest discussion we have in the FEC business meeting.
Should you need more info or want to give input, please contact me.

## How to deal with the hardware diversity hidden behind the same FESA class?

**What we want to avoid... Treatment code!**
In GM, important classes such POW have many treatment codes modifying slightly the behaviour, interface...
This is very complex to manage and we want to define a way to develop FEC software that does not require such a mechanism.

Nonetheless, we need a way to communicate to the high-level control the differences in the devices.

### The problem

Here, we don't want to give an exhaustive list of the cases solved with treatment codes but more an introduction to explain what we need to solve. For example of classes where treatment codes are used look at the bottom of this page.

The problem can be split in two parts: the acceptable values for a given instance and the available services given by a device.

#### Differences in acceptable values

Here we talk about **Control Parameters** (aka Settings). Settings can be continuous (most of the numeric settings) or discrete (mostly enumerated settings and few numerics).

A continuous setting has min/max values that depend on the instance. Normally, those min/max are static and rarely change for a given instance. Few devices have min/max values that can change run-time (e.g. depending on other parameters) or that are given by reading the hardware directly.

Enumerated parameters have a statically defined type (Enum). It happens that, for some devices, not all the possible values in the enum are supported. Again, this is normally something static and only few devices, where all the complexity of the hardware needs to be exposed, have a dynamic range. Bit-patterns (Acquisition & Control) can have different interpretation on different devices but this very low-level representation should be reserved for expert and not shown to the operation.

#### Differences in available services (missing parameters...)

The devices being always attached to a class, the properties (services) available in a device are defined by its class. In some cases where we want to mininise the number of classes and reuse most of the code, we are in a situation where some devices don't support some of the properties defined by their class. For example, for some power converters, there is no control on the state (ON/OFF) or no control on the reference current.

We often meet that problem when we try to hide behind a generic interface a lot of hardware of different types.

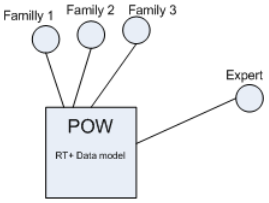
### The possible solutions

Here are several possible solutions to the problem given above. We think the first one is the best and we therefore detail it a bit more.

#### Facade approach

This approach recognises the need to reduce the number of FESA classes and the wish of the equipment specislists to reduce the amount of code they have to manage. The main drawback is that there is not room for dynamic ranges as in the second approach.

The idea is to have to possibility to have several interfaces to the same FESA class. The main FESA class defines the RT behaviour and the data model. It can also have an expert interface and define enumerations that are generic for all devices. The interfaces are the operationnal view of the devices and we can have as many interfaces as we have variant of devices.




The interfaces define the services available for the given family and, for enumerated parameters, the enumeration with only the supported values (and not all the values supported by other families). The definition of min/max can be done statically in the DB on an instance basis.

With this approach and provided there is a good support for it at the FESA tool level, it doesn't cost a lot to the equipment specialist to add a new variety of device and the minimum interface is visible by the operation team without complex logic in between.

When it comes to device grouping, the current constraint is to put in a device group devices that belong to the same FEC class. With the facade approach, we need more flexibility since two power converters can have different facades since they have different possibilities.

By default, the system will name the group headers automatically when all the layouts in the group have a common name e.g. 4 families where the 1st column is called Current. Otherwise the system will put a qeustion mark (or some kind of marker) and the system configurer will be able to specify the name for that specific group. Of course, several layouts can be specified and holes for not applicable parameters will be allowed

| | Cuurent | Acquisition | Status | Unit |
|---|---|---|---|---|
| **PowFam1** | 10.0 | 10.2 | ON | A |
| **PowFam2** | | | ON | |
| **PowFam3** | 10.0 | 9.85 | | A |

#### Meta item approach

A more complex but perhaps more flexible solution to the first problem would be to give to the FEC the responsability to give the acceptable value.

1. The devices publish the acceptable values.

The proposal is to consider all items in a property as objects and therefore to have fields in those objects.
For example, there is a property *Kick* with two fields *current* and *delayed.*

- *Current* is a double value with a continuous range and has the fields
  - value
  - min
  - max
- *Delayed* is a enum value with a discrete range and has the fields
  - value
  - range

Min and max fields are used to describe continuous settings and the range field is used to describe discrete settings (numeric or enumerated). Other fields could be foreseen (for example unit) when this info has a dynamic behaviour.

At FESA level, we should have the possibility to create an item of type Continuous or Discrete setting and in the code the item's fields should be accessible in the usual C++ way.

At JAPC level, the CWM fields should be sorted accordingly if the CMW is not able to transport structures.

#### All OK approach

There is yet another solution that could be applied for missing or reduced services. The FESA class could simply accept everything defined but only react on values valid for the specific instance.

1. The devices accept all the values and don't react on bad ones possibly going back to the previous.
2. The devices accept all the values and go to the closest when the given value doesn't make sense.

This is felt as quite dangerous since it gives a wrong image of the hardware and furthermore the closest state might be different for two different communities.

## Treatment code example

### POW

Each 4-bit Code has a standard meaning:

- 0: not available
- 1: available (standard case) for Actuation: On/Off/Stby/reset
- 7: available but not on knobs
  - STATRM (bit 0 .. 3) status acquisition (off/stby/on/no flt/ up/ remote/nowarn/noIntlk) 2=On-Stby; 3=BFA, 4=CcvOnly; 5=PsbMps; 7=NoKnob
  - CCATRM (bit 4 .. 7) actuation control : 2=On-Stby; 3-On-Off-Reset; 4=CcvOnly; 7:NoKnob
  - AQNTRM (bit 8 .. 11) acquisition of AQN: nbr of AQNs; 6=Dble batch, 9: PpmAqn
  - CCVTRM (bit 12 .. 15) control of CCV: Nbr of CCVs; 6=Dble Batch; 7=ActOnly

*Dble batch*: 2 pulses of same ccv amplitude
*ActOnly*: knob controls actuation, displays actuation & ccv (->reference converter: the CCVs are provided by 'slave' devices depending e.g. on a DEST line)
*CcvOnly*: no on/off (on/off by common 'redresseur'): Corrections Basse Energie PSB
*BFA*: No stby, 15mn actuation delay->ON
*PsbMps*: special status bits
*PpmAqn*: acq in ppm despite CCV is not.

STAQ has 6 different treatment cases for decoding bit patterns.

### POWP - Isolde

Power (AB/PO) type: **AQNTRM** property (1-4:beamLine, 10-17:separator)

### RFPS

CCATRM (1..6) define if aqn (1..10) & ccv(1..n) are meaningful
1 :GAP VETO 2:CCV,CCV1,CCVD1 3 :CCV 4: CCV,CCVD1,MODE,VETO
5: CCV,CAVITY,SYNTH 6: ENABLE,REFACT

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **type 1** | —— | —— | ccv2 | ccv3 | —— | —— | —— | aqn2 | aqn3 | —— |
| **type 2** | ccv | ccv1 | —— | —— | ccvd1 | aqn | aqn1 | —— | —— | aqnd1 |
| **type 3** | ccv | —— | —— | —— | —— | aqn | —— | —— | —— | —— |
| **type 4** | ccv | —— | ccv2 | ccv3 | ccvd1 | aqn | —— | aqn2 | aqn3 | aqnd1 |

### PTIM

- CCATRM: Read PTIM module type. (TRM)
- 4 different Treatment cases for TRAIN

### DIGIO/DIGCTL

**AQNTRM** and **CCVTRM** read the aquisition, resp. the control treatment code. A zero value means the corresponding action is not possible. The different values >0 serve to control the presentation of the data on the workstation level.
If TRM = 18, the CCV value is inverted (all bits inverted)
For DIGIO 28 different Treatment cases currently exist for CCV/AQN decoding of bit patterns.
For DIGCTL, 28 different Treatment cases currently exist for CCV/AQN decoding of bit patterns.

### AIOX

CCVTRM (Type of DAC module)
AQNTRM (ADC present?)

## 1 Comment

**Jakub Pawel Wozniak**

My question is even more basic -> why do we have this hardware diversity behind one fesa class???

Shouldn't we have exactly one class per one device type? We can use some kind of inheritance or linking to achieve the goal.

I think we are solving a wrong problem here...