A Light for Science

From SPEC to Abstract: The Energy Scan Example

From SPEC to Abstract: The Energy Scan Example
Where do we start from

Current status – everything controlled in SPEC:

- sequence
- hardware

What is needed:

- keep the input (element, edge)
- use any sequencer (including SPEC)
- reuse the hardware already controlled in MXCuBE
- make abstraction of the specific institute/beamline hardware or software.

From SPEC to Abstract: The Energy Scan Example
What to do

How to achieve it:

- An abstract hardware object (AbstractEnergyScan)
  - describes the generic sequence (do_energy_scan), using the TaskUtils (cleanup, error_cleanup classes, task decorator)
  - defines hardware object parameters get/set functions and the set of parameters needed (**undulators**, safety_shutter, energy, …) .
  - defines as abstract methods all the actions needed by the sequence (get_static_parameters, **calculate_und_gaps**, calculate_mca_roi, move_undulators, **open_fe**, close_fe, move_energy, find_attenuation…), using the python abc (Abstract Base Classes) module.

```
@abc.abstractmethod
@task
def open_fe(self):
    """
    Open the front end
    """
    pass
```

From SPEC to Abstract: The Energy Scan Example
What to do (2)

- An ESRF specific hardware object (ESRFEnergyScan)
  - gets the specific for the ESRF beamlines hardware object parameters.
  - fills in the abstract methods with ESRF specific actions (get_static_parameters, calculate_mca_roi, **open_fe**, close_fe, fluo_detector_in, fluo_detector_out, find_attenuation).

*mxcollect.xml: <object href="/feshut" role="**undulators**" />*

```
@task
 def open_fe(self):
       sellf.bl_ctrl.undulators.openShutter()
```

From SPEC to Abstract: The Energy Scan Example
What to do (3)

- A beamline specific hardware object (ID29EnergyScan)
  - fills in the abstract methods with the beamline specific actions:

*mxlocal.xml: <undulator> <**type**>u35u</type> <gap_index>0</gap_index>*
<undulator> <type>u21d</type> <gap_index>1</gap_index>
*mxcollect.xml: <commant type="spec"name="**calc_gap**">_ae_wwave</command>*

```
@task
def calculate_und_gaps(self,energy):
    i=0
    for n_und in sellf.bl_cnfg.undulators:
        und_gaps[i]= self.execute_cmd("calc_gap", n_und.getProperty("type"), energy)
        …
         I += 1
    return und_gaps
```

From SPEC to Abstract: The Energy Scan Example
What is next

General Discussion - how to organize the abstraction?

Today we have

- AbstractMultiCollect
- AbstractEnergyScan

Wishlist

- AbstractAttenuation
- AbstractHardware – FE, shutters, motors
- AbstractDetectors – diffraction, fluorescence
- What else…