The European Synchrotron

mxcubeweb
for developers

Collaborating since 2005

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Contents

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Introduction
and
Working in the project

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

About us

Daniele De Sanctis

Antonia Beteva

Olof Svensson

Axel Bociarelli

Jean Baptise Florial (EMBL Gr)

Loic Huder

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Collaboration and partners

- Project started in 2003 at ESRF and became a collaboration in 2005
- Which makes this MXCuBEs 20th birthday :)
- Today we are 15 collaborating partners !

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Collaboration and partners

- The aim is to provide a platform for sharing solutions and know-how
- We are striving towards making MXCuBE and easy to deploy and use application (and extend)
- During my 10 years a very friendly and collaborative spirit with a solution oriented mindset.

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Collaboration and partners

- A very big thanks to all of you !

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

A bit of history

You probably already know the story

- First based on Framework2, General ESRF UI Qt3 Framework
- Framework2+ mostly used for MXCuBE Qt3
- Framework2+ ported to Qt4 (much later Qt5)
- MXCuBE web project started
- HardwareRepository previously part of Framework2 becomes mxcubecore

So, yes !
There are still some very old code in mxcubecore, you have or will probably notice :)

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Organisation

- Steering committee
- Scientific committee
- Developers committee (That’s us :))

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Conventions and good practice

- Like in most projects there are some conventions and good practices

- We try to base those on what's widely used in the software community (it makes tooling easier and minimizes cognitive overhead ;))

- Document in the CONTRIBUTING.md file *https://github.com/mxcube/mxcubecore/blob/develop/CONTRIBUTING.md*

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Brief summary

- Bugs, use the GitHub issues. Check for duplicates and provide as much information as possible
- Docstring are written with google style doc-strings and we have been using sphinx for generating documentation *(not used since a long time but now back working again, thanks Fabien :))*
- Code style is PEP8, we are using flake8 (still in progress)
- Black for formatting

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Recommendation

- Consider using a editor with good support for formatting, linting and testing
- vscode, pycharm, (emacs or vi if one still is adventurous)
- There are the .vscode setting .editorconfig files committed to help setting up the environment
- Use conda or similar tool to handle your virtual environment and to install development dependencies like flake8 and pytest

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Github actions based CI

- PyTest
- Test coverage report
- Lint (still needs some work)
- Tag and publish package to PyPi
- Build of documentation (*still work in progress*)
  - https://mxcubeweb.readthedocs.io/en/latest/
  - https://mxcubecore.readthedocs.io/en/latest/

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

mxcubecore
for developers

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

We never really use mxcubecore on its own but you can, for fun (or for testing)

Checkout repository

https://github.com/mxcube/mxcubecore.git

Instructions: https://github.com/mxcube/mxcubeweb

Setup your environment

Favourite editor, Pytest, flake8, black, pre-commit

(conda-environment-dev.yml contains the development libraries/tools needed)

Run the tests

Simply running pytest

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

What is mxcubecore - A control system agnostic library for sharing common routines and integration of instrumentation

These objects, routines and instrumentation interface logic, is implemented as HardwareObjects

Let's have a look at some code:

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Simple Example

```
import os
from gevent import monkey
monkey.patch_all(thread=False)

import mxcubecore
from mxcubecore import HardwareRepository as HWR

ROOT_DIR = os.path.abspath(os.path.dirname(mxcubecore.__file__))

hwr_config_path = "%s%s%s" % (
    os.path.join(ROOT_DIR, "configuration/mockup"),
    ":",
    os.path.join(ROOT_DIR, "configuration/mockup/test"),
)

print("Configuration path:")
print(hwr_config_path)

HWR.init_hardware_repository(hwr_config_path)

def print_value(val):
    print(val)

HWR.beamline.resolution.connect("valueChanged", print_value)
HWR.beamline.resolution.get_value()
HWR.beamline.resolution.set_value(1.4)
```

- **Gevented**
- `HardwareRepository` is accessed as `HWR`

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Simple example - output

```
DEBUG:HWR.DiffractometerMockup.moving motor {motor} to position {position}
+=================================================================================================+
| role             | Class                 | file                          | Time (ms)| Comment
+=================================================================================================+
| beamline         | Beamline              | beamline_config.yml           | 100      | Start loading contents:
| machine_info     | MachineInfoMockup     | mach-info-mockup.xml          | 37       |
| transmission     | TransmissionMockup    | transmission-mockup.xml       | 2        |
| energy           | EnergyMockup          | energy-mockup.xml             | 1        |
| beam             | BeamMockup            | beam-mockup.xml               | 4        |
| flux             | FluxMockup            | flux-mockup.xml               | 268      |
| detector         | DetectorMockup        | detector-mockup.xml           | 5        |
| resolution       | ResolutionMockup      | resolution-mockup.xml         | 1        |
| safety_shutter   | ShutterMockup         | safety-shutter-mockup.xml     | 2        |
| sample_changer   | SampleChangerMockup   | sample-changer-mockup.xml     | 9        |
| diffractometer   | DiffractometerMockup  | diffractometer-mockup.xml     | 738      |
| sample_view      | SampleView            | sample-view-mockup.xml        | 2        |
| mock_procedure   | ProcedureMockup       | procedure-mockup.yml          | 103      |
| beamline         | Beamline              | beamline_config.yml           | 1278     | Done loading contents
+=================================================================================================+
```

The beamline object can now be accessed via `HWR.beamline` (as seen previously)

But how does this work (in a nutshell :)) ?

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

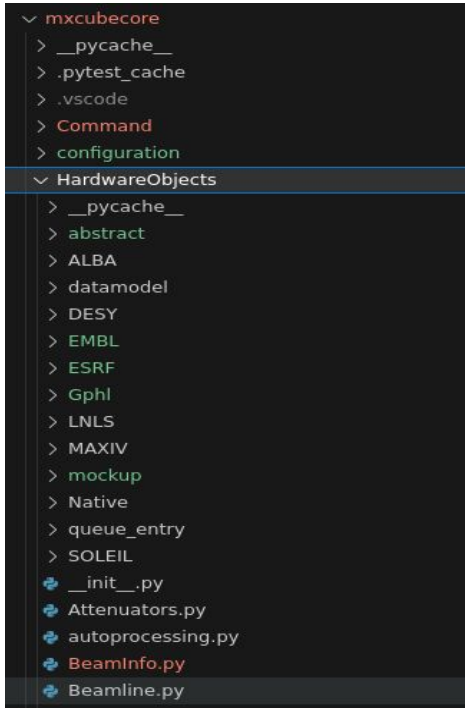
Loading HardwareObjects

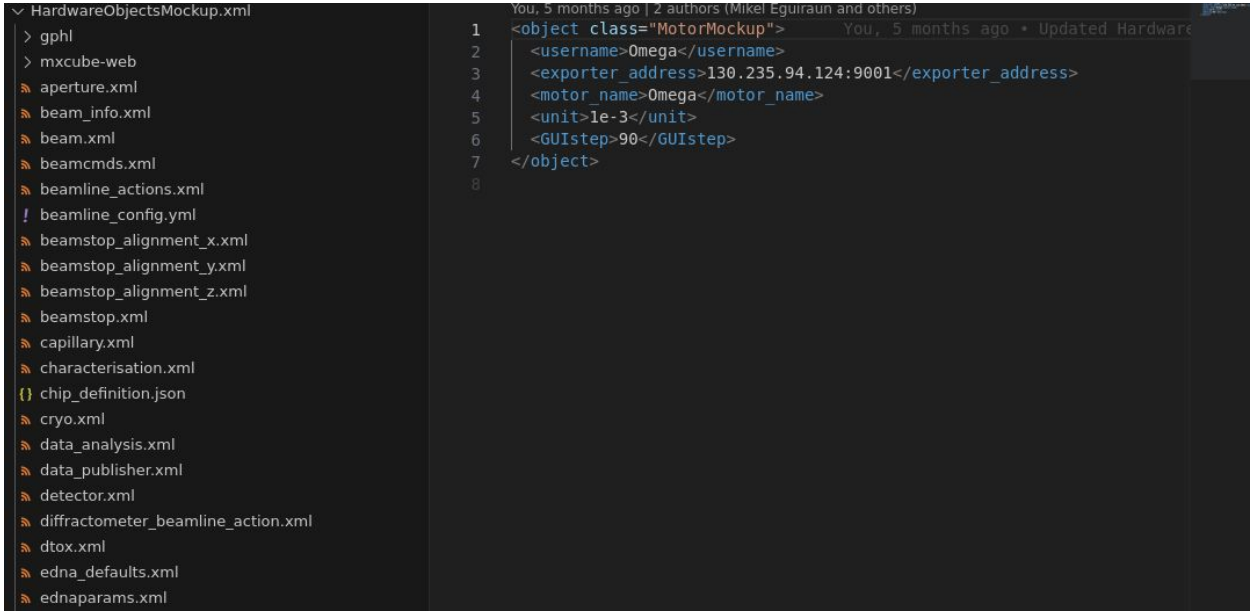
```
DEBUG.HWR.DiffractometerMockup.moving motor {motor} to position {position}
+=========================================================================================================+
| role             | Class                  | file                          | Time (ms)| Comment          |
+=========================================================================================================+
| beamline         | Beamline               | beamline_config.yml           | 100      | Start loading contents:
| machine_info     | MachineInfoMockup      | mach-info-mockup.xml          | 37       |
| transmission     | TransmissionMockup     | transmission-mockup.xml       | 2        |
| energy           | EnergyMockup           | energy-mockup.xml             | 1        |
| beam             | BeamMockup             | beam-mockup.xml               | 4        |
| flux             | FluxMockup             | flux-mockup.xml               | 268      |
| detector         | DetectorMockup         | detector-mockup.xml           | 5        |
| resolution       | ResolutionMockup       | resolution-mockup.xml         | 1        |
| safety_shutter   | ShutterMockup          | safety-shutter-mockup.xml     | 2        |
| sample_changer   | SampleChangerMockup    | sample-changer-mockup.xml     | 9        |
| diffractometer   | DiffractometerMockup   | diffractometer-mockup.xml     | 738      |
| sample_view      | SampleView             | sample-view-mockup.xml        | 2        |
| mock_procedure   | ProcedureMockup        | procedure-mockup.yml          | 103      |
| beamline         | Beamline               | beamline_config.yml           | 1278     | Done loading contents
+=========================================================================================================+
```

- The configuration files are parsed and the hardware objects loaded
- Starting with the “beamline object” and traversing the “hierarchy” downwards loading the children
  - yaml files loaded via `HardwareRepository.load_from_yaml`
  - xml files loaded via `HardwareReposiotryClient._load_hardware_object`
- And yes, our plan is to (eventually) replace XML with YAML !

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

HardwareObjects in action

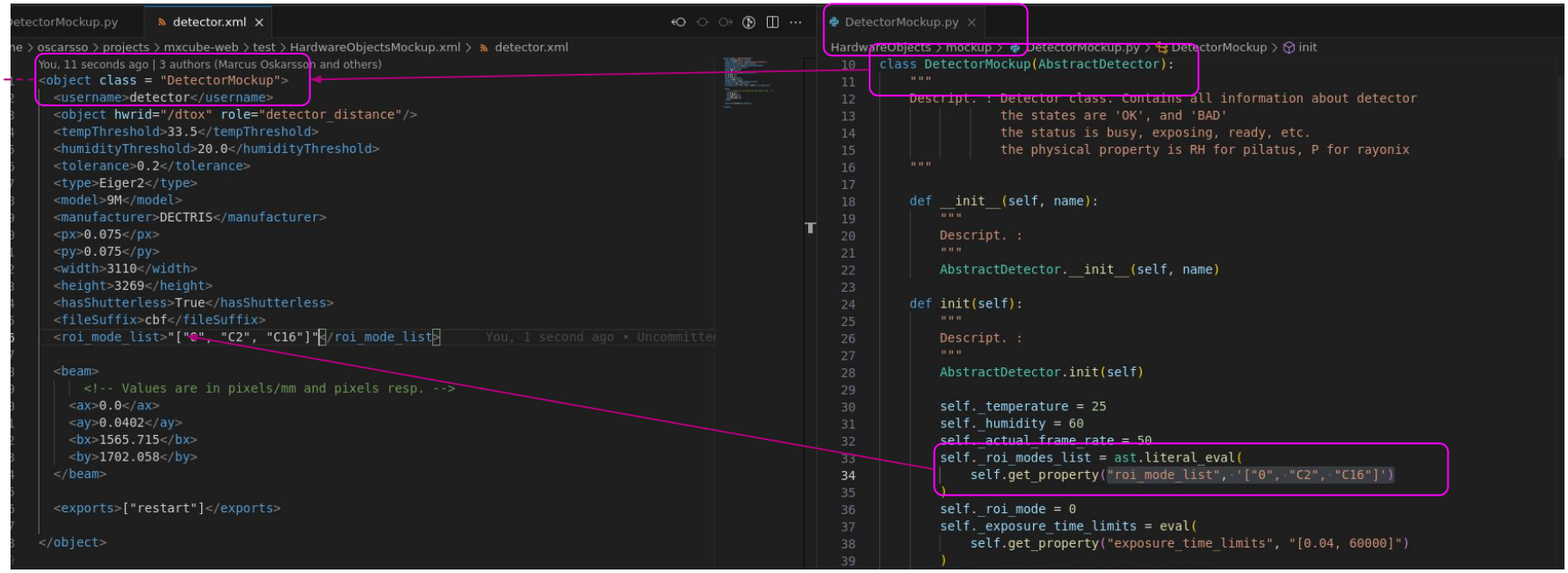




- Each site has its own directory for site specific HardwareObjects and code
- As you now know there is a folder with yaml and xml file configuring these objects

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

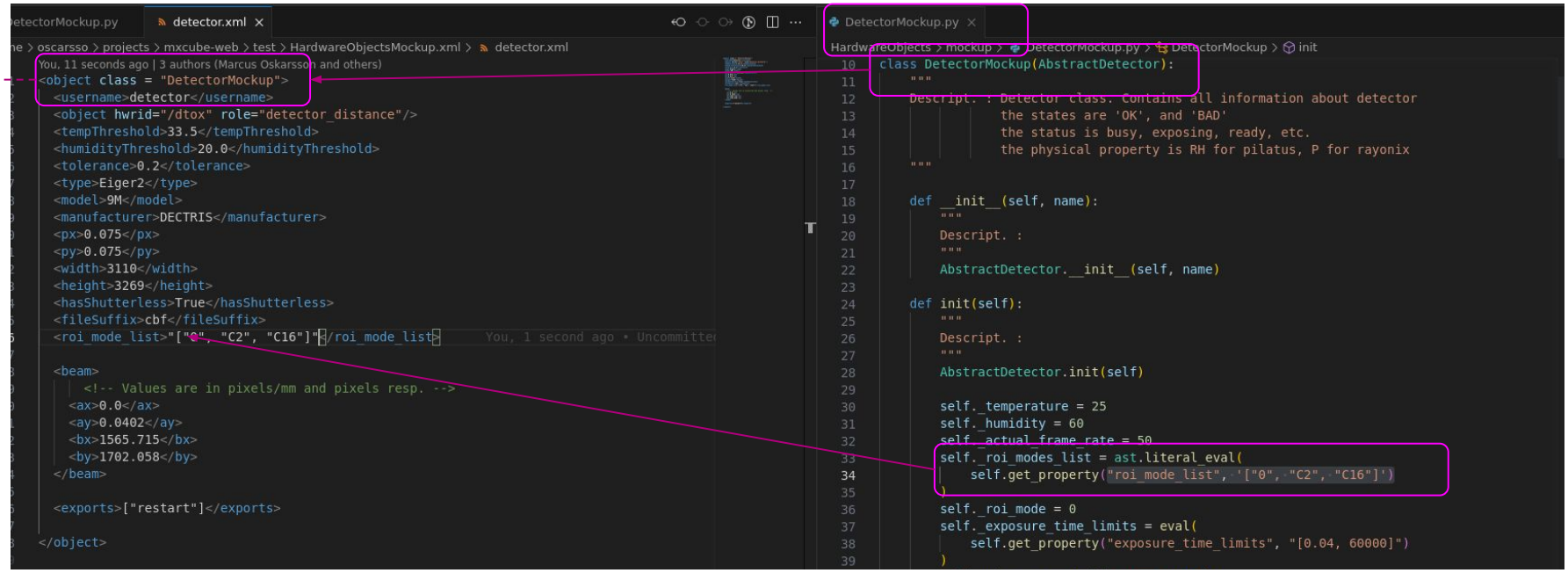
HardwareObjects in action



- Important: **class name and file name should be the same**
- XML is **not validated** against any schema, we often use **ast.literal_eval to evaluate python structures** (There are some ideas on how to provide a stricter definition of the configuration for each object)
- You may see **Equipment and Device** instead of **Object** as well (those are **Equipment and Device are deprecated** and should be replaced by **Object !**)

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

HardwareObjects in action



- values in <start_tag></end_tag> are retrieved with **get_property("tag")**
- objects with a role are retrieved by **get_object_by_role("role")**
- ***Future development: (might get replaced by simply adding those roles/attributes directly when object is parsed)***

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Channels and commands

HardwareObjects can communicate with control systems via something called channels and commands

```
class Microdiff(MiniDiff.MiniDiff):
    def init(self):
        self.phiMotor = self.get_object_by_role("phi")
        self.exporter_addr = self.get_property("exporter_address")

        self.update_state(HardwareObjectState.READY)      You, 5 months ago • Possibili

        self.x_calib = self.add_channel(
            {
                "type": "exporter",
                "exporter_address": self.exporter_addr,
                "name": "x_calib",
            },
            "CoaxCamScaleX",
        )
        self.y_calib = self.add_channel(
            {
                "type": "exporter",
                "exporter_address": self.exporter_addr,
                "name": "y_calib",
            },
            "CoaxCamScaleY",
        )
```

Channels and commands provide an abstraction for the various control systems used

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Channels and commands

Channels and commands provide an abstraction for the various control systems used

Supported are: Exporter (EMBL), Tango, Taco, Tine, Sardana, EPICS, (SPEC), **BLISS (Not via channels and commands)*

- *Channels for values (and events)*
- *Commands for "functions"*

HardwareObjects inherit CommandContainer meaning that we can use add_channel and add_command

- The library pyDispatcher is used for handling signals/events
- Each protocol implements a CommandObject, a ChannelObject and protocol specific "Client" for handling events and other protocol internals.
- A channel emits a "update" signal with the new value on a event

Antonia will talk more about this tomorrow

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Briefly about HardwareObjects

- **The mxcubecore work have harmonised the API we use for these HardwareObjects: (summary of changes:** https://github.com/mxcube/mxcubecore/blob/develop/changelog.txt

- ***There are still some work to be done for instance with signals and further refining the interface of certain objects and converting to YAML***

- A base class for all Hardware Objects **HardwareObject** and has a well set of state; READY, BUSY, OFF, FAULT etc.

- A set of commonly used subclasses derived from HardwareObject;

- Introduction of **BeamlineObject** for facilitating access to “well known” HardwareObjects

- Yaml configuration

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Briefly about HardwareObjects

- The basic objects we use are, HardwareObject, AbstractActuator,
- From these there are a number of objects derived that everybody most likely will use or can reuse: AbstractNState, AbstractMotor, AbstractTransmission, AbstractResolution, AbstractDetector, AbstractShutter, AbstractSlits, AbstractBeam and … actually the list is quite long :)
- To keep in mind:
  - HardwareObject provides set_state and get_state
  - AbstractActuator provides set_value, get_value and set_limits, and get_limits
  - AbstractNState is a AbstractActuator where the value is in finite set of values (states)
- **Thanks to Antoina: https://github.com/mxcube/mxcubecore/blob/develop/Hierarchy.pdf**

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

About the queue

(Ok, take a deep breath ;))

HardwareObjects are important for instrument control, but WE want to collect data !

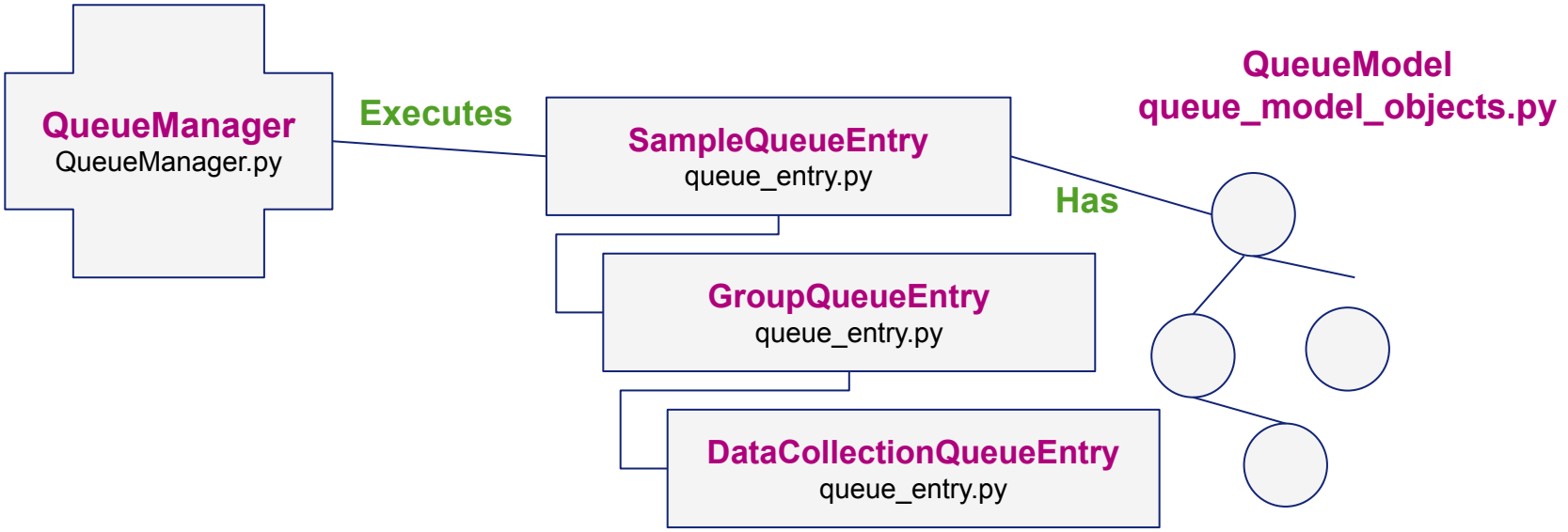
Data acquisition tasks/protocols are implemented as QueueEntry objects

These QueueEntryObjects are executed via a queue (QueueManager)

The queue further provides means for automation

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

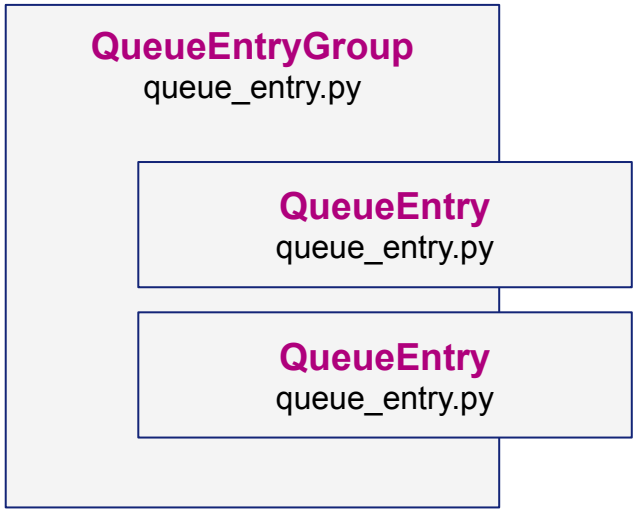
Original Queue - Static



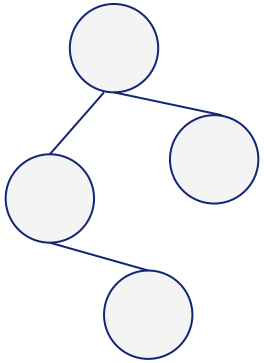
- **QueueManager** executes **QueueEntry** objects each having data models.
- Each **QueueEntry** in turn is a tree with Parent and child nodes, where the tree is traversed downwards (depth first), children gets executed after the parent
- Each entry has a **pre and post execute** that gets executed **before and main execute function**
- In **practice maximum three levels of QueueEntry** objects are used, what's called **SampleQueueEntry, GroupQueueEntry** and **QueueEntry**. (Workflows sometimes uses more) ***(The use of Group might be removed/changed)***

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Current Queue - Static



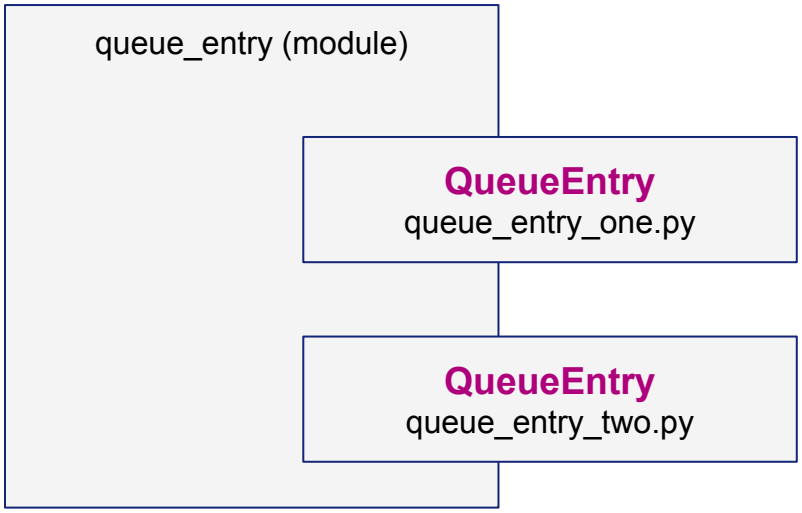
QueueModel
queue_model_objects.py



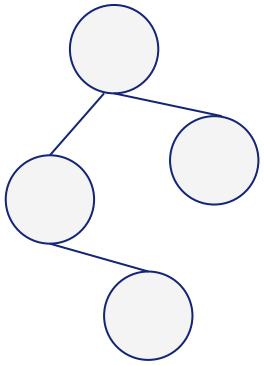
- QueueModels are written as python objects that gets converted to dicts when passed around, hard to debug and know what to pass.
- Creation of each type of Queue entry is more or less manually wired for creation in the UI

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

New Queue - Dynamic



CommonQueueModels

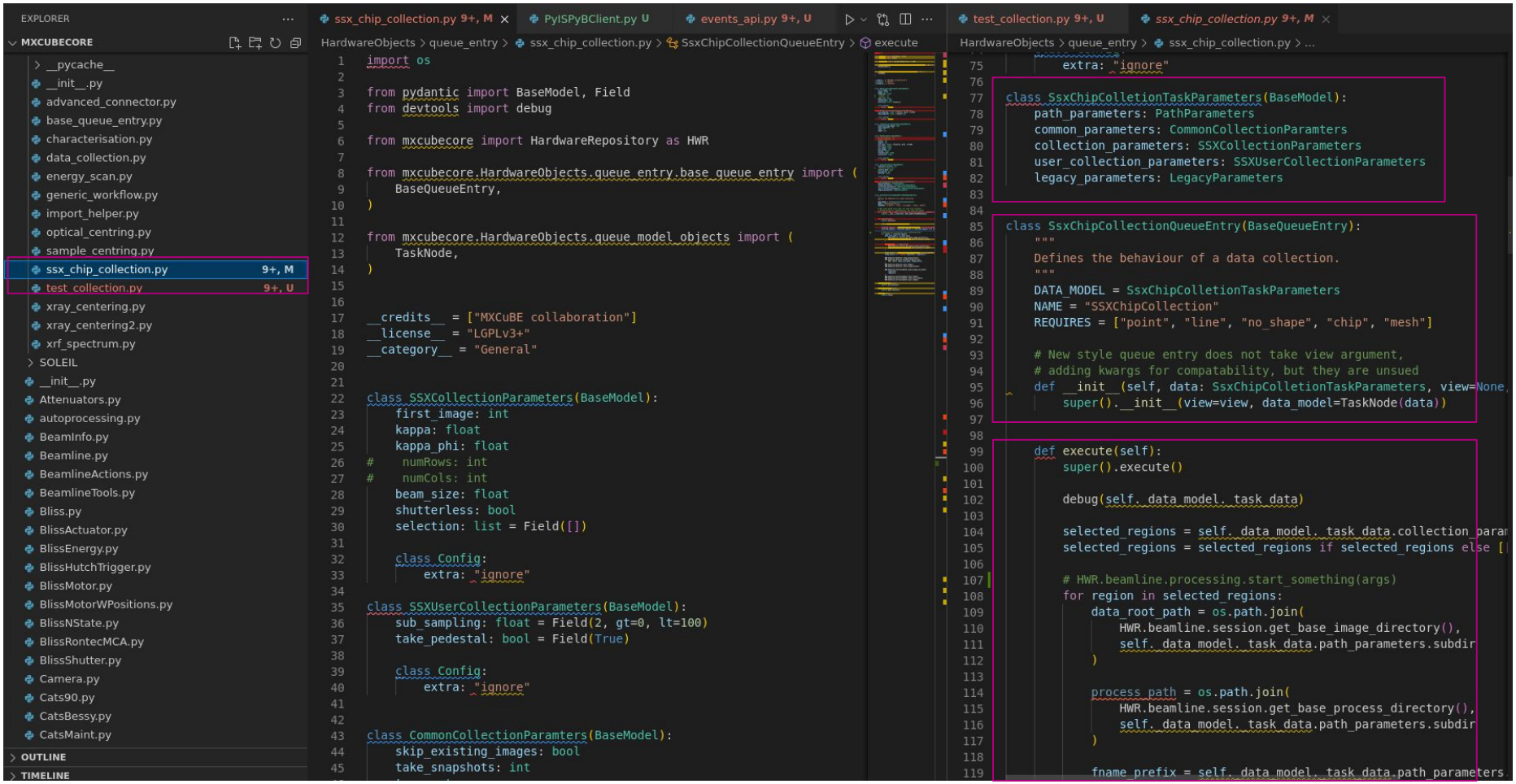


- Each QueueEntry is self **contained in its own file with its specific models**
- Models are expressed as **Pydantic models**
- All common models are **shared in common queue models module**
- Each QueueEntry object in the queue_entry module (or other search path) is **dynamically imported**
- **A generic UI interface/dialog** can be built based on the **model data associated with a QueueEntry** (an evolution of the current workflow dialog)

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

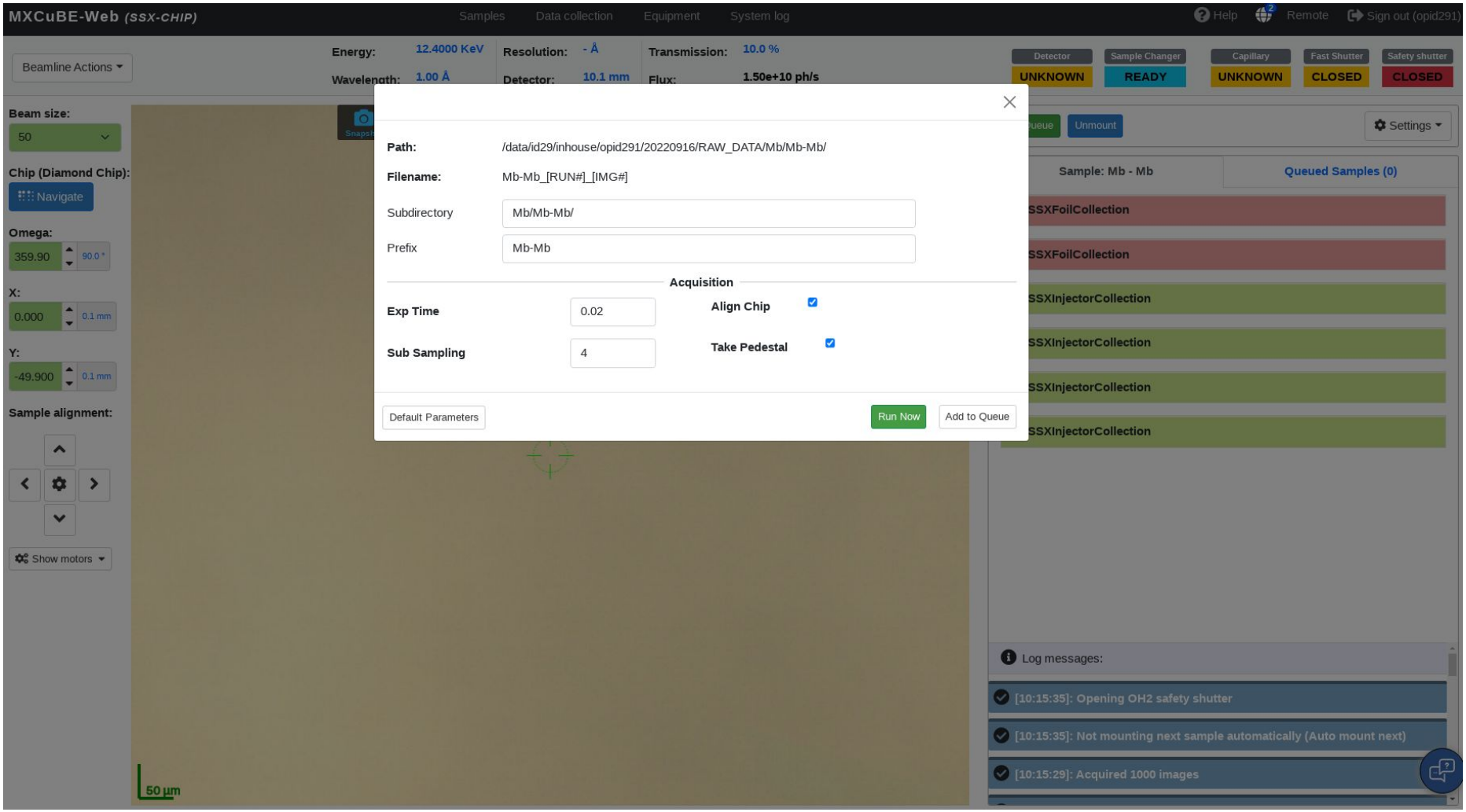
New Queue Proposal

Example - New style queue entry



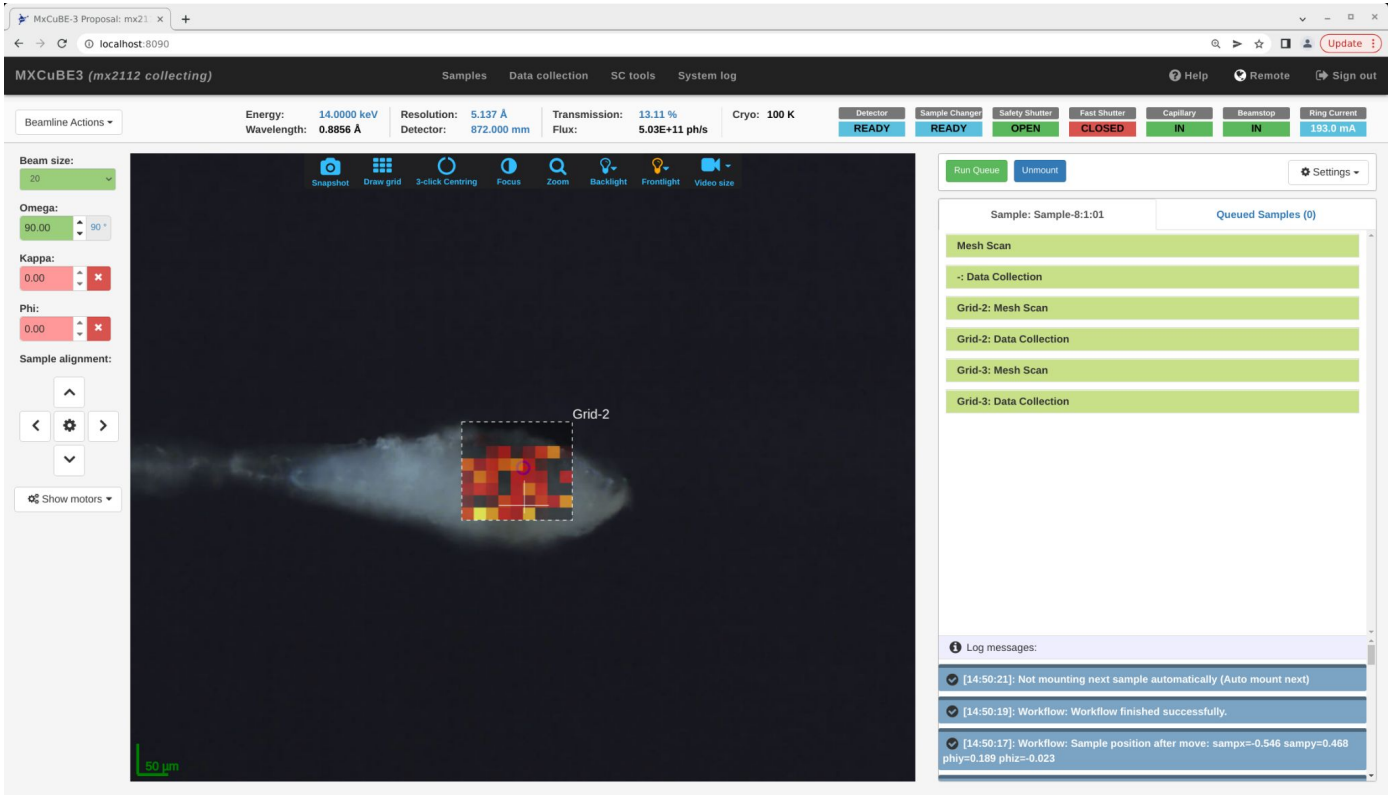
Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Example UI



Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Using mxcubeweb



Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Working in the project

- Same conventions and guidelines as mxcubecore
- CI Pipeline is more or less the same as mxcubecore
- Additionally running Cypress end to end (e2e) tests
- Something called ESLint for javascript
- Build of documentation (*still work in progress*)
  - https://mxcubeweb.readthedocs.io/en/latest/

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Installing and configuring

Instructions on: https://github.com/mxcube/mxcubeweb

```
(mxcubewebtest) oscarsso@laposcarsson:/tmp/mxcube$ mxcubeweb-server -r /tmp/mxcube/mxcubeweb/test/HardwareObjectsMockup.xml/ --static-folder /tmp
/mxcube/mxcubeweb/ui/build/ -L debug
```

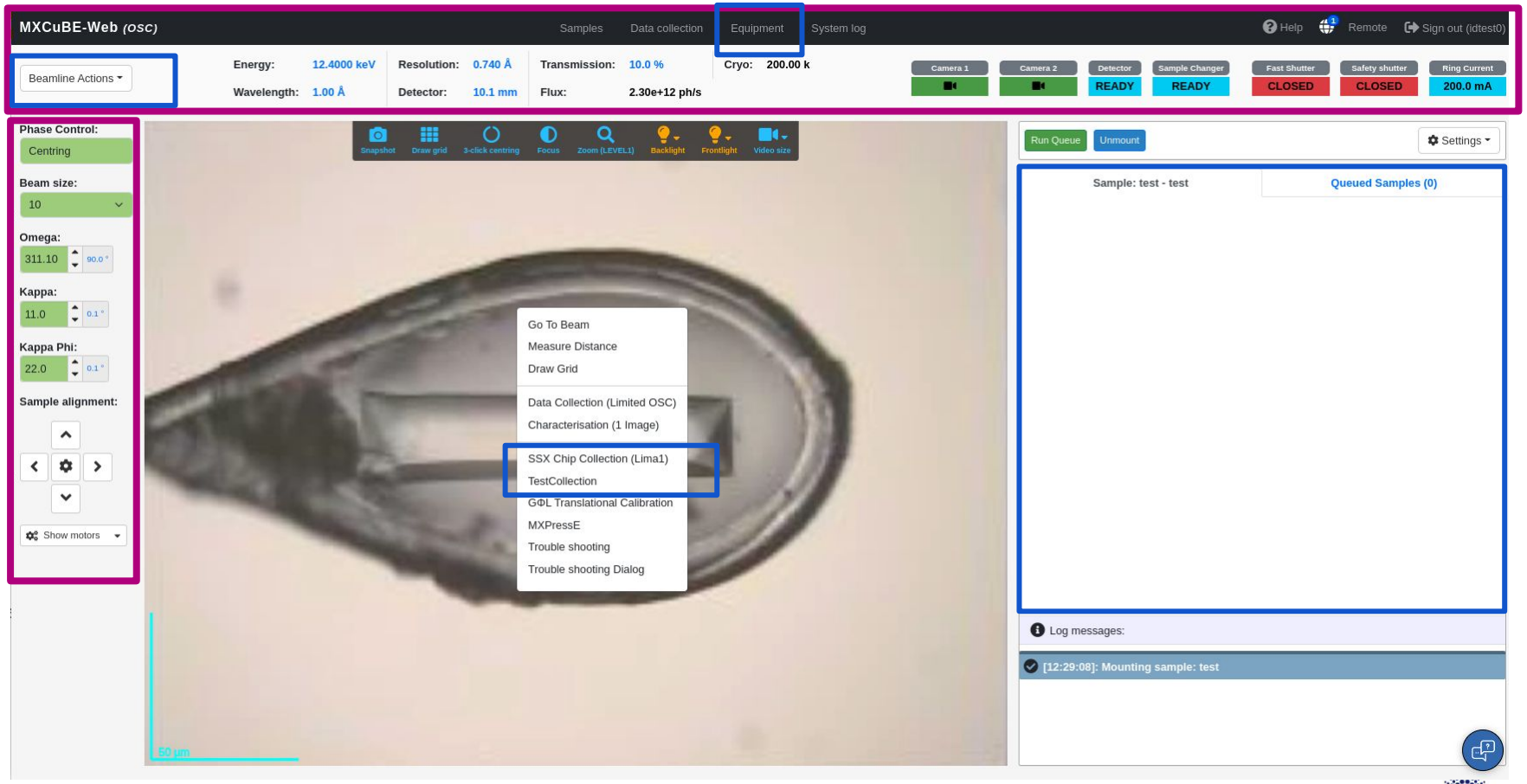

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Installing and configuring

Instructions on: https://github.com/mxcube/mxcubeweb

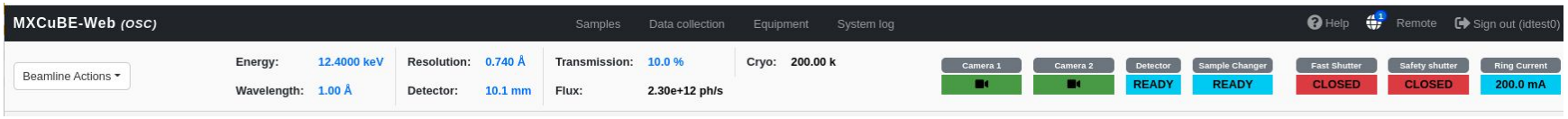
Designed for MX experiments - with the idea of same interface on all sites

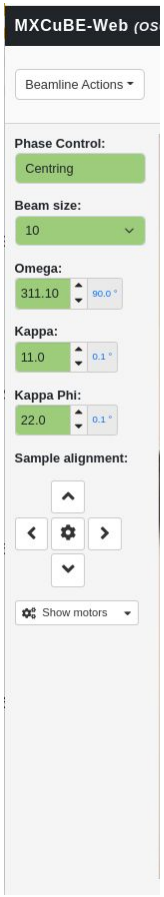
To a certain extent configurable instrumentation control, procedures / methods



Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Installing and configuring





The display of available instrumentation is configurable in ui.yaml

To the left motor control and on the top “beamline setup”

```yaml
sample_view:
  id: sample_view
  components:
    -
      label: Omega
      attribute: diffractometer.phi
      role: omega
      step: 90
      precision: 2      Marcus Oskarsson, 20 months ago • I
      suffix: °
    -
      label: Kappa
      attribute: diffractometer.kappa
      role: kappa
      step: 0.1
      precision: 1
      suffix: "°"
```

```yaml
beamline_setup:
  id: beamline_setup
  components:
    -
      label: Beamstop
      attribute: beamstop
    -
      label: Capillary
      attribute: capillary
    -
      label: Fast Shutter
      attribute: fast_shutter
    -
      label: Safety shutter
      attribute: safety_shutter
    -
      label: Detector
      attribute: detector
    -
      label: Energy
      attribute: energy
      step: 0.001
      precision: 4
      suffix: keV
```

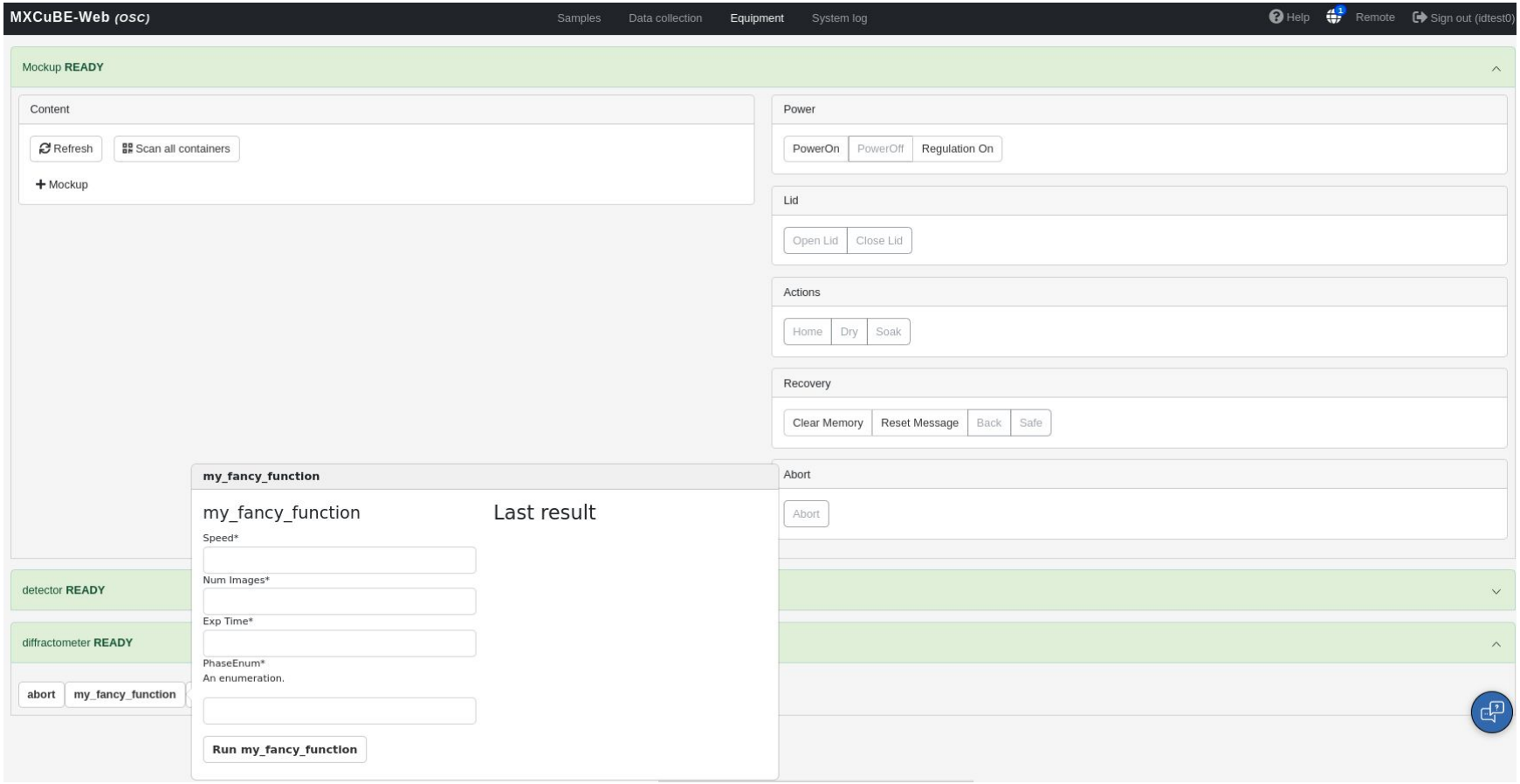

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Methods and procedures can be added in three ways:

- Equipment view - For not so often used or temporary instrumentation commands
- Beamline action - For procedures that are frequently used and involves more than a simple command
- Queue entry / task - For collecting data

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Equipment view - For less often or temporary instrumentation commands



Methods are automatically added if they are "exported" with the export tag and the method is type hinted (at least with a return type)

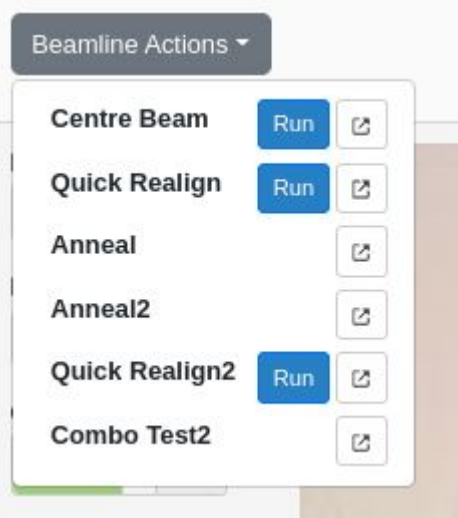
```
<exports>["abort", "status", "my_fancy_function", "my_other_funny_function"]</exports>
```


Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Installing and configuring

Beamline action - For procedures that are frequently used and involves more than a simple command

Configured as the beamline_actions of the Beamline hardware object




```
<object class="BeamlineActionsMockup">
    <commands>[
      {"type": "controller", "name": "Centre Beam", "command": "HardwareObjects.mo
      {"type": "controller", "name": "Quick Realign", "command": "HardwareObjects
      {"type": "controller", "name": "Anneal", "command": "HardwareObjects.mockup
      {"type": "annotated", "command": "HardwareObjects.mockup.BeamlineActionsMock
      {"type": "annotated", "command": "HardwareObjects.mockup.BeamlineActionsMock
      {"type": "annotated", "command": "HardwareObjects.mockup.BeamlineActionsMock
    ]
    </commands>
</object>
```

The controller commands define the command arguments programmatically via the CommandObject.add_argument method

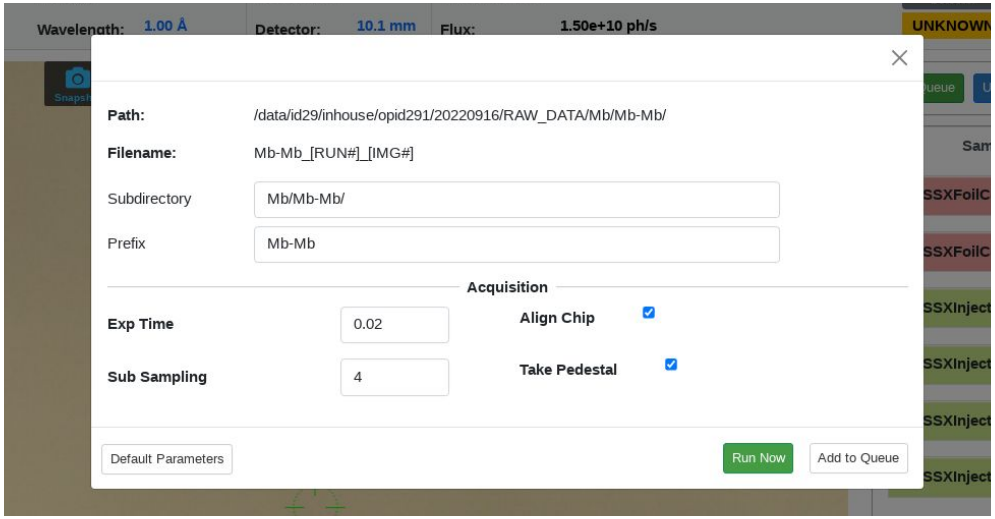
The annotated command uses the method typehints to define the arguments *(still work in progress, sorry the UI display is currently broken, we are working on it)*

Mikel will make a practical about Beamline actions

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Installing and configuring

Queue entry / task - For collecting data



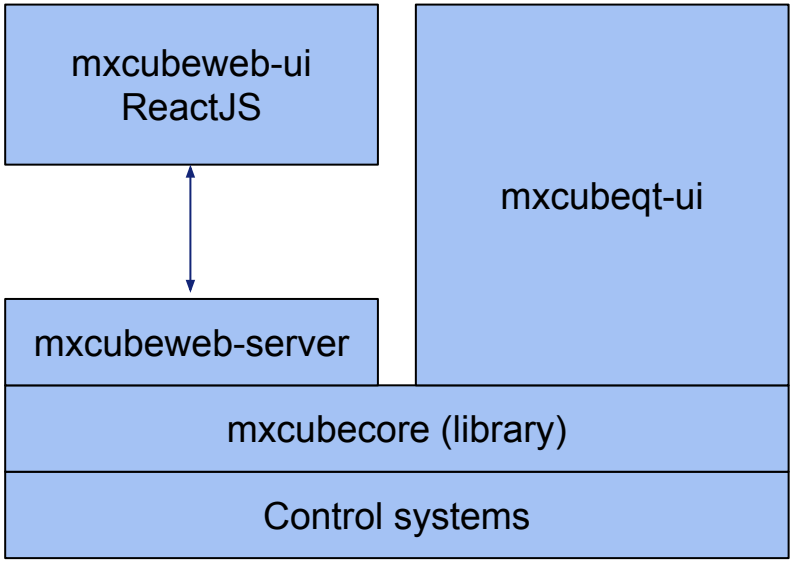
Write a task that takes a Pydantic model and add it to available_methods of Beamline object

```
82      legacy_parameters: LegacyParameters
83
84
85  class SsxChipCollectionQueueEntry(BaseQueueEntry):
86      """
87      Defines the behaviour of a data collection.
88      """
89      DATA_MODEL = SsxChipColletionTaskParameters
90      NAME = "SSXChipCollection"
91      REQUIRES = ["point", "line", "no_shape", "chip", "mesh"]
92
93      # New style queue entry does not take view argument,
94      # adding kwargs for compatability, but they are unsued
95      def __init__(self, data: SsxChipColletionTaskParameters, view=None,
96          super().__init__(view=view, data_model=TaskNode(data))
97
98
99      def execute(self):
100         super().execute()
```

```
available_methods:
    datacollection: True
    characterisation: True
    helical: True
    xrf_spectrum: True
    energy_scan: True
    mesh: True
    ssx_chip_collection: True
    gphlworkflow: True
    test_collection: True
```


Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Overview



Frontend built using Javascript, React and bootstrap, redux …)

Asynchronous communication done over websockets

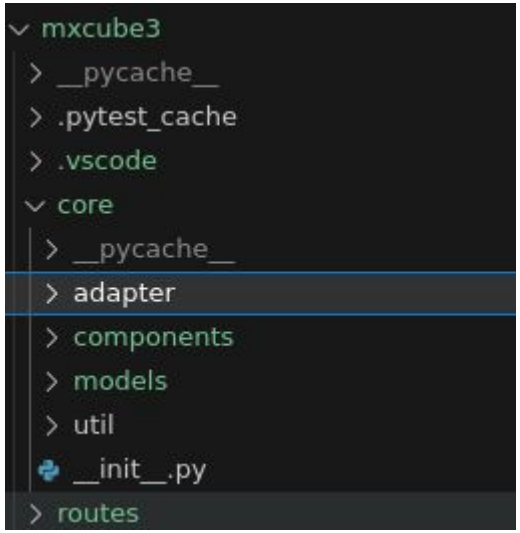
Some of the bigger libraries we are using:

- Flask
- Flask security for handling user and session
- Flask-SocketIO for web sockets
- SpecTree for OpenAPI documentation, http://localhost:8081/apidoc/swagger/

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Overview

Something called an adapter object converts (adapts) a HardwareObjects to the web world, get and set over GET and POST requests and events over websockets.



Adapters found in the adapter directory, adapts to standardised API of HardwareObjects.

Components are larger pieces of functionality, such as queue, lims, workflow with static mapping to routes. (Some might become adapters in the future)

Models for defining complex data structures and marshaling

Routes contains explicitly defined routes

Marcus Oskarsson (marcus.oscarsson@esrf.fr)

Future perspectives

After MXCuBE Web 4

- Update/Modernize Javascript code
- Using typescript ?
- Exchange Speectree for FlaskOpenAPI3 or other ?
- Possibly remove dependency on gevent
- Your ideas

Marcus Oskarsson (marcus.oscarsson@esrf.fr)