MXCuBE Developers' meeting

November 8, 2022

DRAFT

Participants :

Marcus Oscarsson, Antonia Beteva, Oscar Svensson(ESRF)
Rasmus Fogh, Gerard Bricogne (Global Phasing)
Jacob Oldfield , Nicolas L (ANSTO)
Michael Hellmig (HZB)
Meghdad Yazdi, (MAXIV)
Roeland de Boer (ALBA)
Bo Yi (NSRCC)
Leticia C, (LNLS)

Qt / mxcubecore development plans

RF introduced the problem. The mxcubecore develop branch is developing quite fast under the impetus of the MXCuBE web version, and work is now beginning on the (long agreed) refactoring of the queue. No site is maintaining a Qt-interface version in sync with mxcubecore/develop, and different sites are to different degrees quite far behind, having adopted to varying degrees the extensive refactoring that has been happening in mxcubecore. The code at various sites has diverged so much that it is impossible to maintain code that should work at multiple sites without supporting multiple parallel branches and extensive cut-and-paste. A discussion (better: agreement) on a target for integration of the Qt branch with mxcubecore would be indicated. Which changes should be included, in the first instance? Which program versions (of Python, Qt and other) should it be based on? How do we plan to get there, with the available resources? And how do we fit it into the agreed branch and release procedures?

It is noted that several developers have been lost from the project (at EMBL Hamburg and elsewhere) and so resources are stretched thin. Most development of mxcubecore is happening at the ESRF, and since they do not use Qt interfaces they can hardly do much in terms of keeping the connection to the Qt interface alive. RdB has been doing considerable work on bringing the ALBA version up to the tip of the mxcubecore develop branch, but RdB is the only developer on the project at ALBA now, and will not have time to continue with this particular job until 2023. It does not help that mxcubecore is changing so rapidly.

The question is raised which Qt version should be aimed for. Some sites are still on Qt4, Soleil, ALBA (and GPhL) are on/adopting Qt5, but AB proposes going for the current Qt

version – Qt6. [Post meeting: Qt4 is long out of support and hard to install (last release 2011), Qt5 is on limited legacy support only, and Qt6 is already at version 6.3/6.4].

There are discussions on the general approach. Several propose sticking to the tip of the development and updating regularly to keep changes small at each stage. ANSTO for instance has a 3-month release cycle and only does bug fixes in between releases. While this is undoubtedly the best approach, the question is raised on how to achieve it realistically in a situation where many sites are very far behind and find it hard to find resources. One proposal is to agree on a target for Qt integration that is short of the current development tip and/or continuing development of mxcubecore (e.g. refactoring of the queue system) in a separate branch to allow the sites to catch up. To the extent that the refactoring would break compatibility it might anyway be necessary to use a new major version for these changes. This would, however, run a very real risk of different branches diverging so far that they effectively split up the project (as MXCuBE history shows).

It is discussed how well the current release system supports dealing with these problems. The master branch has been stable for a long time, but sites are trying to catch up directly with the develop branch. Also, ESRF production code already uses (a version of) the develop branch. In theory the system of a stable master branch, a develop branch moving ahead, and freedom for individual sites to write site-specific code in any way they wish should be sufficient, However, RF notes that this system is based on a model where only the master branch is released and used. MXCuBE reality is that several sites are doing continuous development from their own, individual starting points not necessarily limited to site-specific code, and the release model does not explicitly consider how to deal with production use of development branch code, or tracking and incorporating changes from side branches. One possibility would be to use tags more extensively (as ANSTO does internally). This would make it clear which starting point you were using in each case but would not run the risk of diverging development that you would get with separate branches (nor would it support local development to the same extent, of course).

The first thing to do should be to take a status of where the various QT-using sites are, in development terms.
ACTION: MO to send out a call for the Qt development status to be included in status reports at the December MXCuBE meeting.

It is agreed that sites that work with Qt interfaces should get together and agree on a plan for the way forward. No action points is raised, so a good time might be at the MXCuBE meeting, when we have heard status reports from the sites.

Continuous Integration

1. JO give a lightning overview of his Continuous Integration PR. It includes using the newest Python system for building and installing code, with the Poetry framework for publishing, and the use of Github actions. One advantages of the system is that it simplifies the handling of dependencies including optional (as in site-specific)

dependencies. The system would work equally well for conda or VM installation. Part of the PR proposes to use the Black code regulariser which, now that Python 2.7 is no longer supported is being moved out of ‘safe’ mode. Likewise the addition of the (very aggressive) Flake8 coding style enforcer. A first run with all checks enabled produced over 1000 errors on the current code base, and changed 800+ files. It is noted that Flake8 is specific to each python version. 1) This means that we should make sure that we are using the right Flake8 at each site. 2) the tests are currently run for all supported Python versions from 3.7 onwards; we should consider if we can narrow that down.

Black/Flake8 would be run prior to making PRs. We might consider making passing a condition for acceptance, but there would be no automatic code changes in the repository.

RF raised the problem that due to the very widespread changes in code, black/flake8 would make it harder to make major merges (as is required if there are independent changes on both branches being merged). As opposed to the case of comparing pre-black with post-black, you would be looking at extensive difference and have a hard time deciding where a given difference arose and how it should be handled. The discussion following this suggested that one could start with quite permissive settings for black/flake8, catching (and fixing) only the most egregious problems, and gradually extending the scope.

ACTION: MO to add questions about dependencies and specifically Python version to his pre-meeting questionnaire.

Automatic documentation

JO had produced an introduction to the problem. It was agreed that autogenerated documentation was a very good idea. The biggest practical problem would seem to be that this requires consistent systematic doc strings – which are not universally present in the current code. We did agree earlier to change over to Google style doc strings, and there was a certain push towards introducing this, but there is clearly a way left to go.
Another question is whether to adopt Sphinx or MKDOCS for the automatic documentation. Sphinx is a bit more established (and is in use internally at ANSTO), but Sphinx uses rst text whereas MKDOCS uses the more familiar markdown language. Sphinx supposedly has a markdown plug-in, but it is untested whether that will work sufficiently well with the entire family of Sphinx tools.

The sense of the meeting is that something like this should be adopted, and that the meeting would welcome a recommendation for the Sphinx/MKDOCS choice from a knowledgable person.

Queue refactoring

The ESRF (MO) have started refactoring the queue system, as long agreed would be desirable. The goals of the refactoring is to introduce Pydantic data classes to precisely define the content of data structures; this allows type hints, better defined information transfer between processes and computers, and better linting than dictionaries. Also to allow for plug-in addition of queue entries, simplification of the mechanisms for adding them, and splitting of the large and unwieldy files that currently hold queue model objects (QMOs), and queue entries. MO pointed out that Pydantic classes were pure data holders (which had also been the original intention behind the QMOs) and that getting rid of the QMOs and moving their active functionality elsewhere, would simplify the system, which was slightly in disarray currently.
The point has had a certain amount of discussion on github, mainly between RF and MO. The conclusions (summarised here) was that adding Pydantic, splitting and moving the main files, and adding capacity for plug-in addition were capabilities that could be added and gradually extended without breakage. RF, however, pointed out that the queue was currently defined as a tree of QMOs, and that changing this (however desirable) would be a breaking change in the way of enqueueing entries that would require prior agreement on the new system. MO proposed to make a MAPP (official enhancement proposal) to start this discussion.

Any Other Bussiness

JO communicated that ANSTO was 90% along the way of making an Ophyd detector simulator. GB hooped that this could be presented as a talk in the December MXCuBE meeting

Next meeting

We will have one more meeting before the December in-person meeting in Grenoble. Date and time TBD