THE QUB LIBRARY

S. Petitdemange, G. Berruyer, BLISS Group, ESRF, Grenoble, France

Abstract

2D detectors are increasingly used as counting devices on our beamlines. Video cameras are also extensively used, and users need interactions between displayed images and the rest of the beamline instrumentation (e.g. graphical selection of scan limits using video images). As a consequence, the need for efficient tools to integrate data from 2D detection systems as part of the beamline control applications is an inescapable challenge. In this development, the creation of tools for manipulating other data types (1D, MCA spectrum…) is naturally included. The QUB library has been developed to fulfil these needs.

SPECIFICATIONS

Users view

One main goal is to provide ESRF users with a common set of tools for data visualisation. The look and feel should be coherent from one application to the other and from one beamline to the other. To provide a homogeneous between data display and beamline control, peripheral tools like drawings, colormap and others should exist and be flexible enough to be added or removed from an application in the simplest manner.

Programmers view

As our applications are based on python/Qt, the choice of a python library using Qt/PyQt as window library is evident. Qwt is our choice for 1D display.

Our graphical applications for beamline control are built using the "BLISS framework". This idea is to create more or less independent graphical components (Bricks) and mix them to produce a beamline application benefitting from the services of a dedicated framework. Using standard components, we can build custom graphical applications for each beamline and QUB data display widgets can be added as "BLISS framework bricks". Following this principle, a feature in the QUB library should be as a tool and the library must provide interfaces to make the different tools easily fit together. As an example, the library should contain a 2D display tool, a data source, managing data from a CCD and a colormap tool to improve visualization of the image. Then programmers, using the corresponding interfaces should be able to join in a very simple manner these three tools in an application.

IMPLEMENTATION: THE MAIN POINTS

We developed the QUB library following the specifications previously described. A lot of tools have been created but also compound tools (tools made with tools) in order to reduce the code to be written by developers: a class has been created to display a simple Qt pixmap (*QubGraphicsView*) but also the class *QubDataImageDisplay*, using *QubGraphicsView*, which takes a data file as input and display its contents providing all necessary tools like print, colormap management, sections.

Generic containers

Generic container widgets (viewers) have been designed to allow a common look and feel of all data display objects of the library. These "viewers" deal with two kinds of objects:

- "Action" which manages displayed data (print, colormap, sections…). They are accessed with icons that appear at the top, bottom or in a context menu of a viewer.
- "Display area": 1D or 2D representation of real data.

These generic widgets provide methods to add "actions" objects in a very simple manner.

The aim of these containers is to fulfil the users need: same tools around the display, same graphical representation.

Drawings

An interface (Drawing Manager) has been developed to manage drawings on top of display widgets. This "Drawing Manager" holds one or more "Drawing Objects" and one "Drawing Event". The "Drawing Object" creates the drawing using Qt tools (Drawline, DrawCircle …). The "Drawing Event" implements the behaviour based on input events (keyboard, mouse). Adding more complex drawings is as simple as writing the code to create the drawing itself using standard Qt objects. Their behaviour and communication with the image have been encapsulated in the library to simplify programmer's life.

Optimization

Basic objects for 1D and 2D representations have been optimized to display images from live video cameras, as well as other sources. We pay particular attention to the resource consumption (memory, CPU time ...) and currently achieve 25 Mpixels/s which is displayed on the screen.

OXIDIS: THE QUB SHOWCASE

The Qub library gives flexibility for beamline control graphical interface development. Users still need a standalone application to visualize the data took on ESRF beamlines. *Oxidis* is a generic, modular application to display the largest amount of data type. The code of this application is, for 90% of it, part of the QUB library.

More than just a data display (1D or 2D), *Oxidis* offers to users all basic tools (print, saving, colormap, sections, zoom ….) and more. New data format, new actions on "viewers", and new calculations between data set can be added to Oxidis using dynamic plugins.