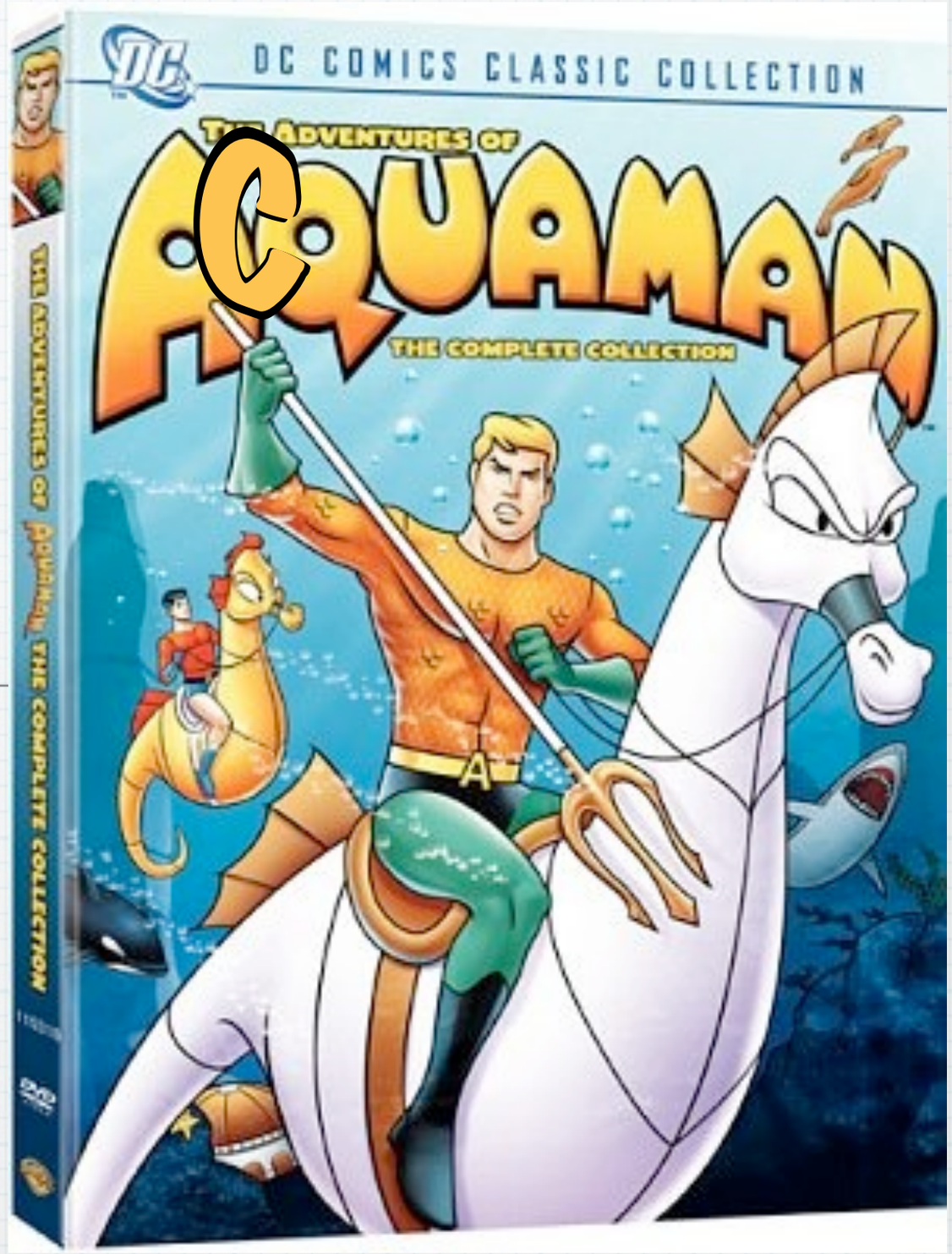
Acquaman

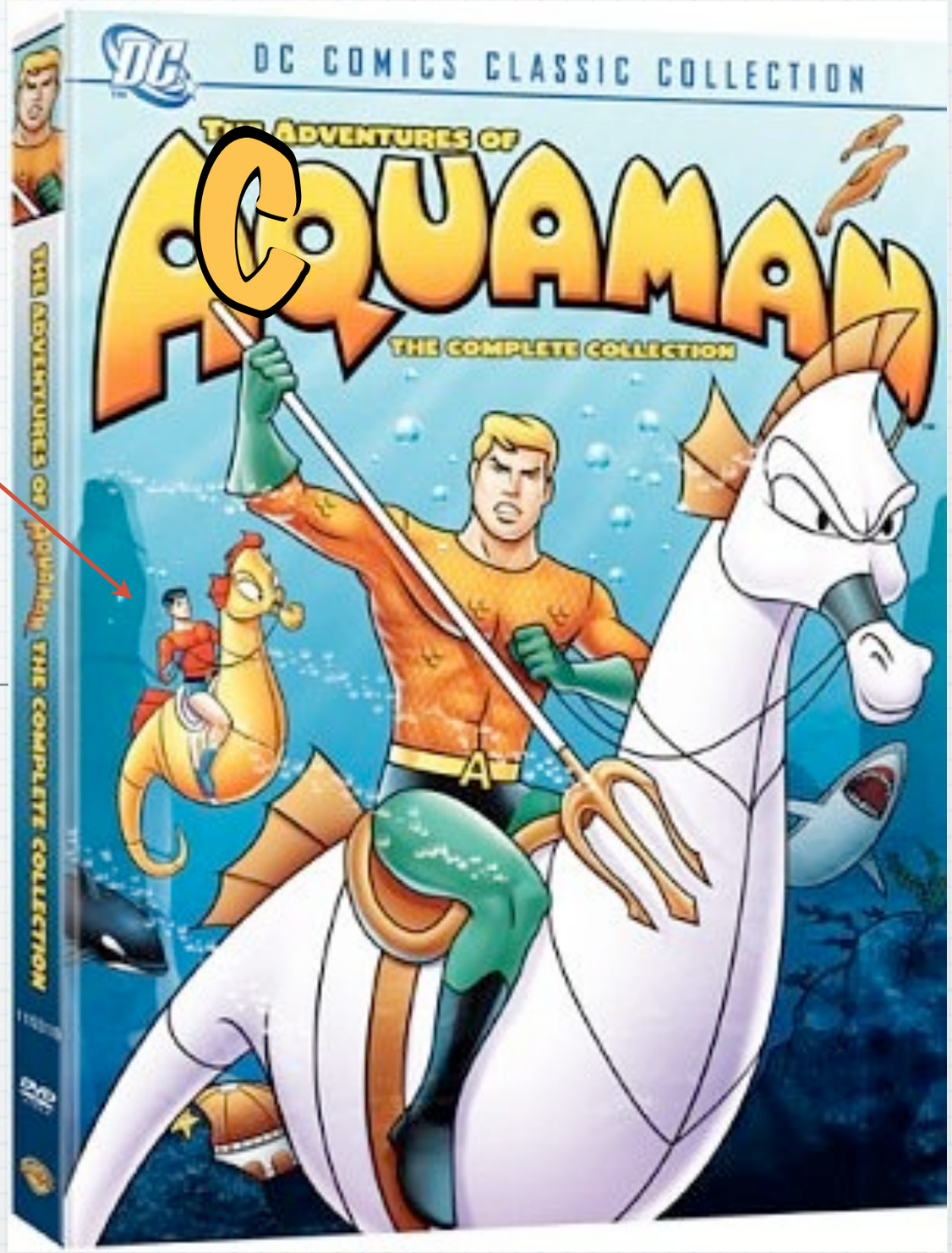
beamline ACQUisition
And data MANagement



Dataman

Acquaman

beamline ACQUisition
And data MANagement



Acquaman is...

a Framework

an Application

Acquaman is...

Controlling Beamlines

Running scans, acquiring data

a Framework

Common user interface components

Visualizing and Plotting (1D, 2D)

Managing users' data (storing, browsing, organizing)

an Application

Acquaman is...

Controlling Beamlines

Running scans, acquiring data

a Framework

Common user interface components

Visualizing and Plotting (1D, 2D)

Managing users' data (storing, browsing, organizing)

well, lots of apps, actually...

an Application

Beamline-specific, integrated apps

Dataman (take-home version)

One-off, quick utilities

Acquaman

OPEN SCANS

EXPERIMENT SETUP

- Emission Scan
- Spectrometer controls

EXPERIMENT TOOLS

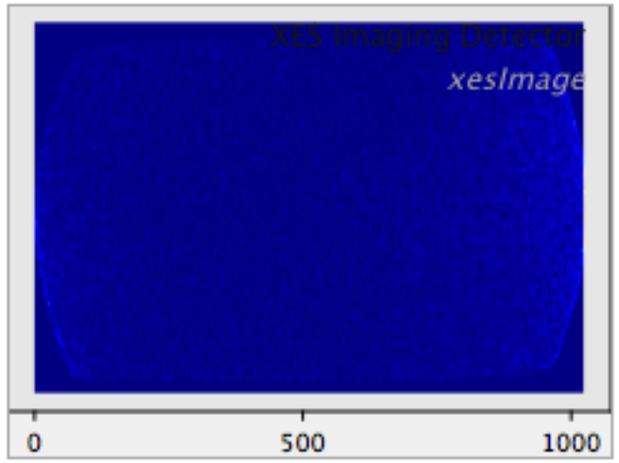
- Workflow

DATA

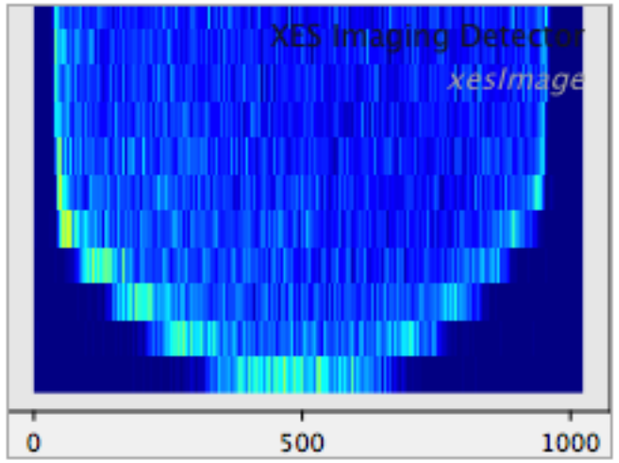
- Runs
  - SGM, Feb 12 - 12
  - REIXS commissioning, F...
- Experiments

mboots' Runs: REIXS commissioning (Feb 28 - Mar 1)

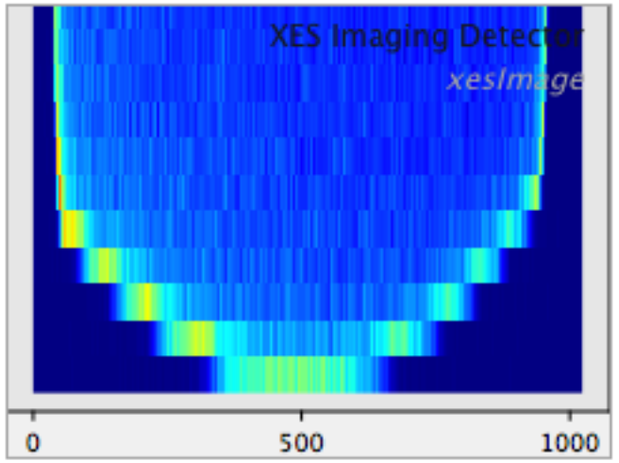
REIXS commissioning (Feb 28 - Mar 1)
Showing all data from this run *(5 scans)*



REIXS XES Scan
Feb 28 at 3:34pm



REIXS XES Scan
Mar 1 at 12:27pm



REIXS XES Scan #1
Mar 1 at 12:34pm

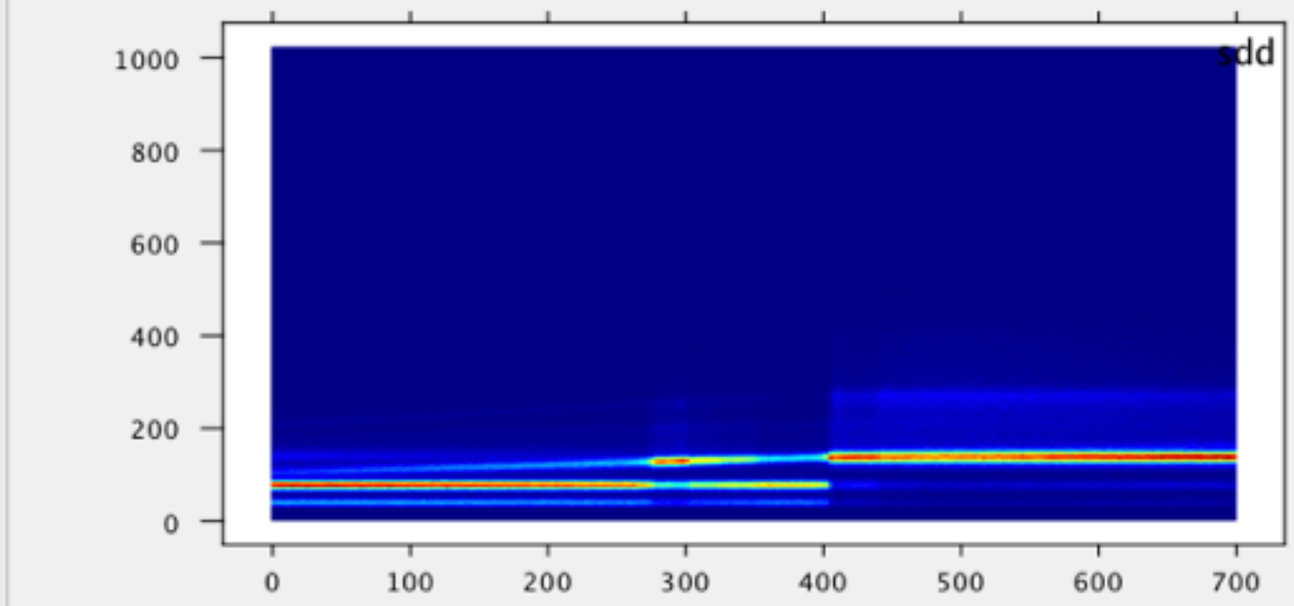
0:00

Beamline-specific, fully-integrated Applications

BEAMLINE CONTROL
SGM Sample Position
SGM Sample Transfer
EXPERIMENT SETUP
Absorption Scan
SGM XAS Scan
EXPERIMENT TOOLS
Workflow
OPEN SCANS
Calcium1_ #2
DATA
Runs
SGM, Mar 2 - 2 (2010)
Experiments

Editing Calcium1_ #2

Status Frame Text



Calcium1_ #2 | tey | tfy | I0 | I0_2 | eV_fbk | ringCurrent | Silicon Drift Detector | tey_n | tfy_n

1 | OP | M-S | M-C

Show at same scale | Waterfall | Amount 0.20

Open Scans
Calcium1_ #2
Mar 2 (2010) at 8:25pm, on [no s...
Open... | Save | Close

Scan Information
name Calcium1_
number 2
date Mar 2 (2010)
time 8:25pm
run
SGM, started Mar

Notes
Grating: Medium Energy
Integration Time: 0.5
Comments:
0.916667

Sample Information
name [no sample]
date
time
elements
Choose a different sample..

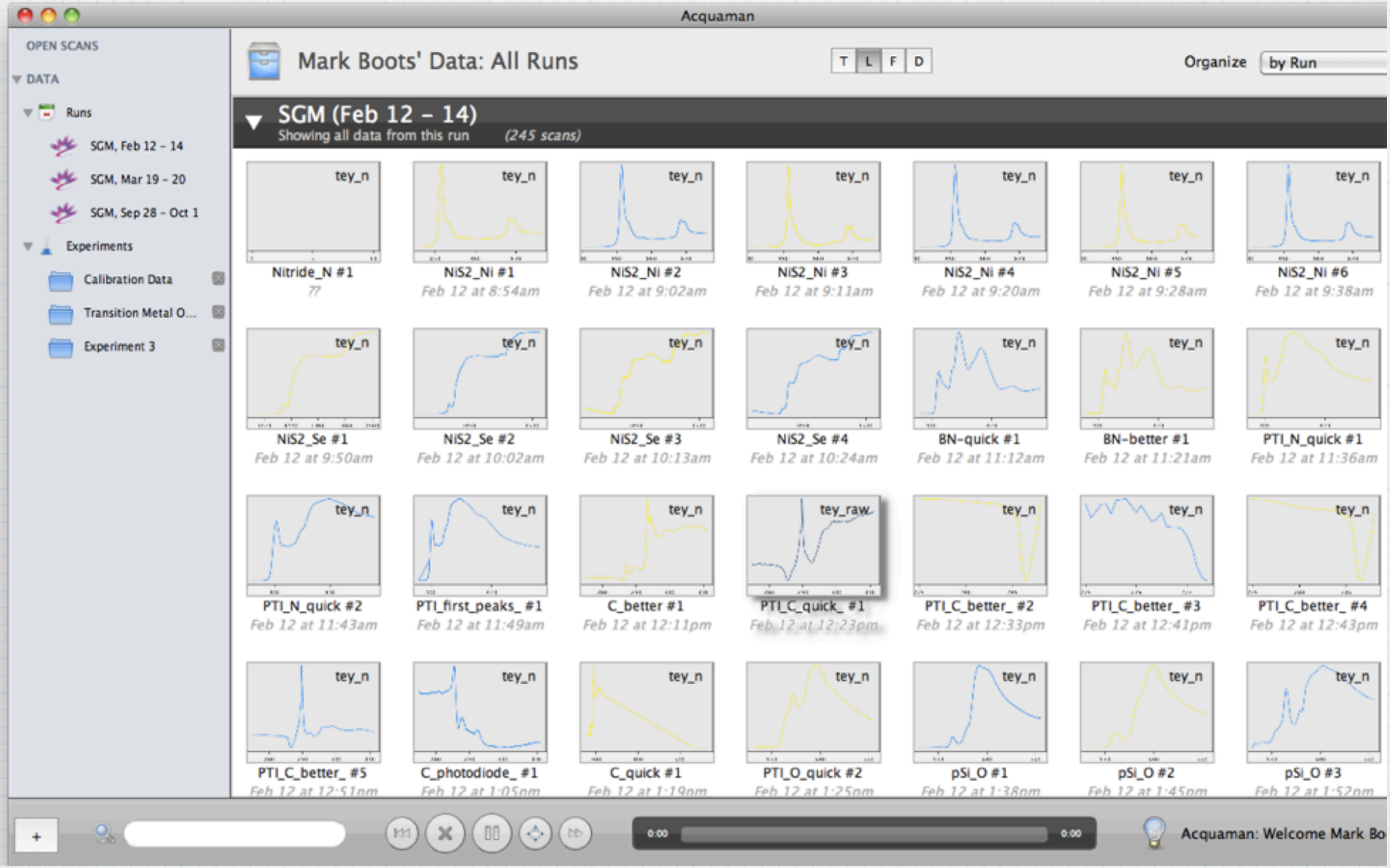
Plot Data
Data Source
Calcium1_ #2
tey (t...
tfy (tfy)
+
name
description

Beamline Information

0:00 0:00

Alert: AMXASScan: There was an e

Beamline-specific, fully-integrated Applications



Dataman
(Take-home visualization, analysis, and data mangement)

Vortex Detector

Single Element Four Element

Regions of Interest

1) Name: Zr Ka1 Energy (eV): 15770

2) Name: Zr Kb Energy (eV): 17660

3) Name: Fe Ka1 Energy (eV): 6403.84

4) Name: Fe Kb Energy (eV): 7057.98

Customize | Remove Selected | Clear List

Sort | Load Defaults

Spectrum

Erase & Start

Stop

Save Spectra

Real Time (s): 100.00

Elapsed Time: 19.12 s

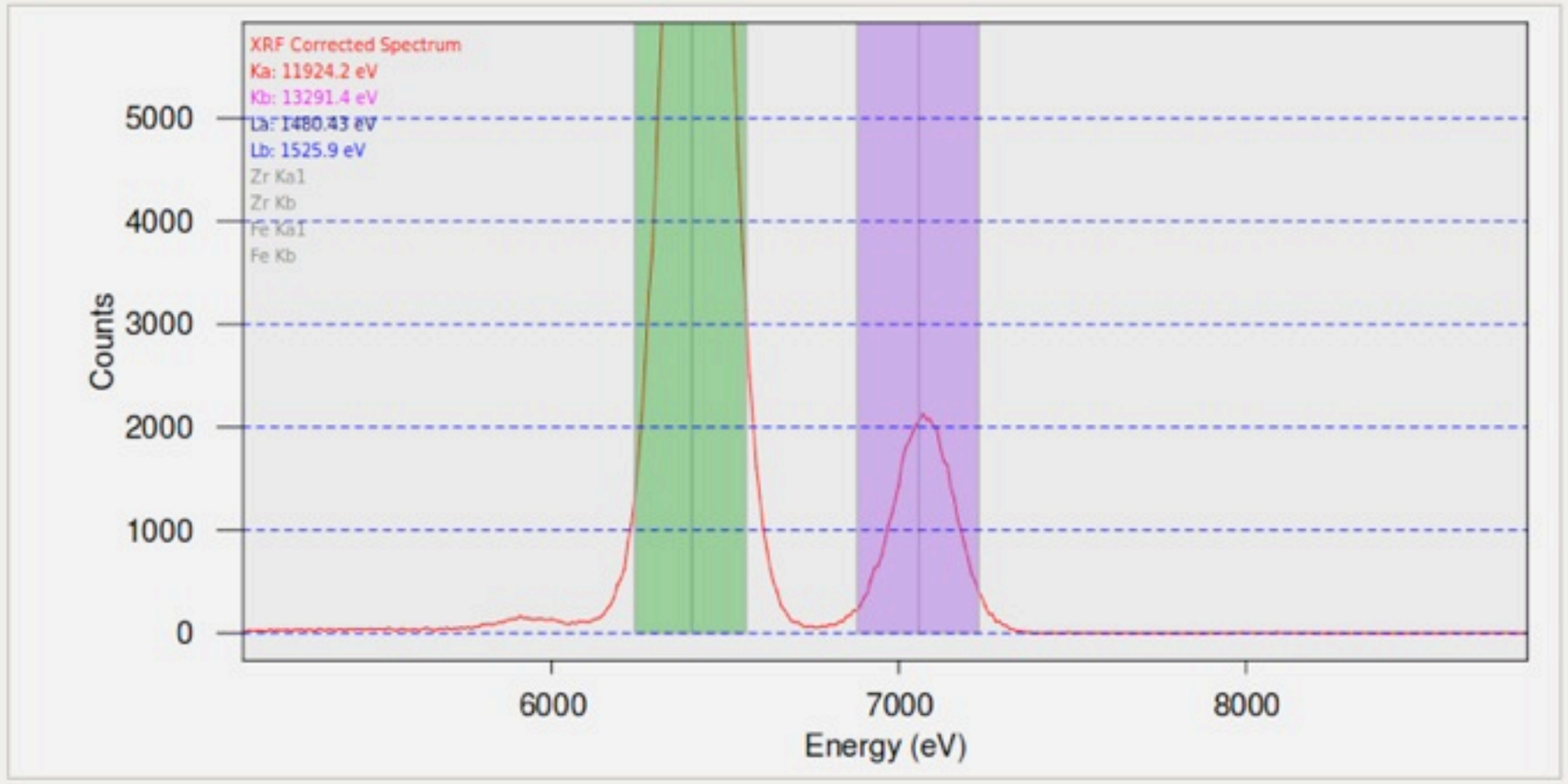
Dead Time: 2.39%

Scale: eV / chan.

Scale: Linear / Log

Max Energy (keV): 20.48

MCA Update: Passive



Add Region of Interest

Sort Elements: Alphabetically / By Atomic Number

Element: Br

Emission Lines: Ka1: 11924.2 eV

Add Custom ROI | Add ROI

One-off utilities for a specific purpose
(Using parts of the framwork)

Why Acquaman?

For Users

for Programmers

Why Acquaman?

Demand for better usability

Provide experiment-specific (!beamline-specific) support

Paradigm shift?

For Users

Tons of data... no data management tools.

for Programmers

Support new techniques, detectors, analysis methods

Enable cross-beamline, cross-facility research

XES on REIXS, iPFY on SGM, etc.

Why Acquaman?

Demand for better usability

Provide experiment-specific (!beamline-specific) support

Paradigm shift?

developed initially for REIXS, SGM, but...

For Users

Tons of data... no data management tools.

Support new techniques, detectors, analysis methods

Enable cross-beamline, cross-facility research

XES on REIXS, iPFY on SGM, etc.

for Programmers

Carefully-designed for reuse

Modular and decoupled

Modern (latest tools, libraries)

Cross-platform

Qt 4, Epics 3.14.12, etc.

Linux
Mac
Windows

From Users: Why can't I...

* Easily plot my emission data from ALS BL8 alongside my absorption data (from the same sample) from SGM?
* Move from PGM to SGM (to repeat the exact same technique...), without having to learn a brand new beamline and totally different software?
* Use one piece of software to manage my entire experiment?

From Users: Why can't I...

* Quickly browse and find a piece of data I took 5 years ago?
* Display a 2D fluorescence map of an unexpected element in my VESPERS data, that I didn't realize was in there when I did the original scan? [The raw data is all here!]

Usability/UCD recap

* Support the experiment, not the machine

1. Support the end-to-end process

Planning | Managing Samples | Measuring | Acquiring Data | Visualization | Analysis | Pondering | Comparing | Browsing old data | Exporting, Graphing | Publication

2. Data management integrated with analysis

3. Do the common (repetitive, expected) tasks exceedingly well

* If a technique is unique to a beamline, support it explicitly and greatly
* If something is common across beamlines, have it look and behave the same across all.

Current Status

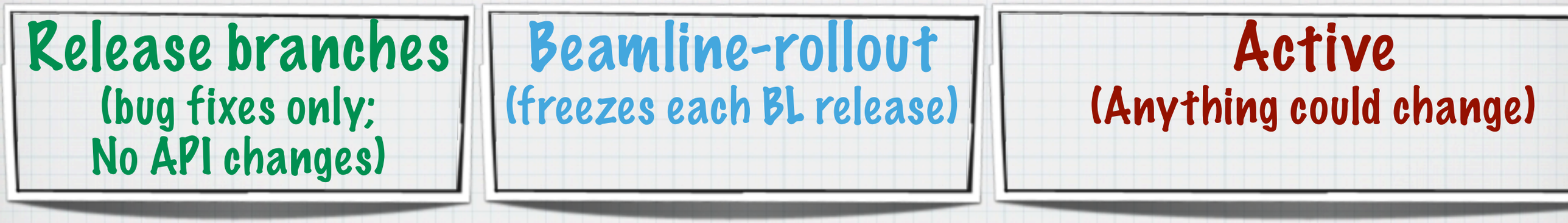
* Haven't hit completeness/usability goals for beamline users yet.
  (SGM, REIXS apps)
* ARE approaching nice usability for programmers

Current Status

* Motivation for opening to other devs:
  * More eyes helps to improve the framework:
    * Find functionality gaps/oversights
    * Bugs
    * Identify confusing APIs, insufficient documentation, etc.
* Therefore, offering to support

Caveats

* Just offering as an available tool
  * (not forcing or pushing for a facility-wide roll-out)
* Still under fast and active development (APIs may change)
  * Mitigate: branches and releases; code ratings (coming soon...)



Goals: Today

1. Identify possibilities for using Acquaman
2. Know how to get it
3. Understand starting points for applying it:
    * Modules and functionality (What can it do?)
    * Overall structure (How does it fit together?)
    * Simple examples (Where do I start?)

Outline

* 0 - Intro (almost done...)
* 1 - Core concepts (from programmer's perspective)
* 2 - Support systems
  * How to get it
  * Source control
  * Documentation
  * Code conventions
* break;

[2 pm]

Outline (cont'd)

* 4. Modules: Functionality and Examples
  * Beamline (moving and measuring stuff)
  * Dataman (Representing and storing user data)
* break; [3 pm]
  * Acquisition (coordinating and running scans)
  * Visualization (Plotting and comparing)
  * Analysis Chains
* Open discussion / Wrapup / Hacking session [4 pm]

Core Concepts

Core Concepts

* 1. Technologies
  * C++
  * Qt
* 2. Paradigms
  * Object Oriented Programming
  * Model/View Programming
* 3. Tools
  * Signals and Slots [Qt]

C++

“ C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as an intermediate-level language, as it comprises a combination of both high-level and low-level language features. It was developed by Bjarne Stroustrup starting in 1979 at Bell Labs as an enhancement to the C language and originally named C with Classes. It was renamed C++ in 1983. ”

Just Kidding!

Qt

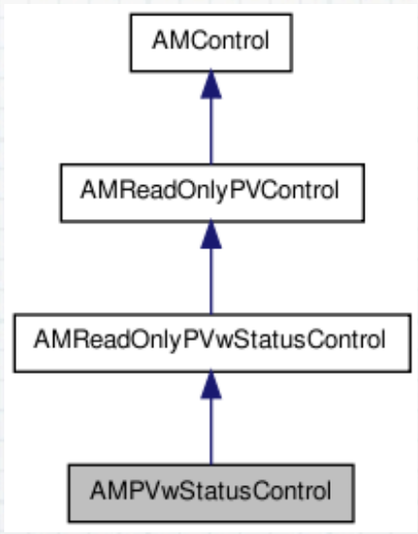
* A C++ cross-platform framework for GUIs and applications
* Offers considerable support for everything from widget creation and files access to animation and threading
* Uses Meta-Object Compiler (moc) to replace callbacks with Signals and Slots
* Currently using Qt 4.7 (up to date)

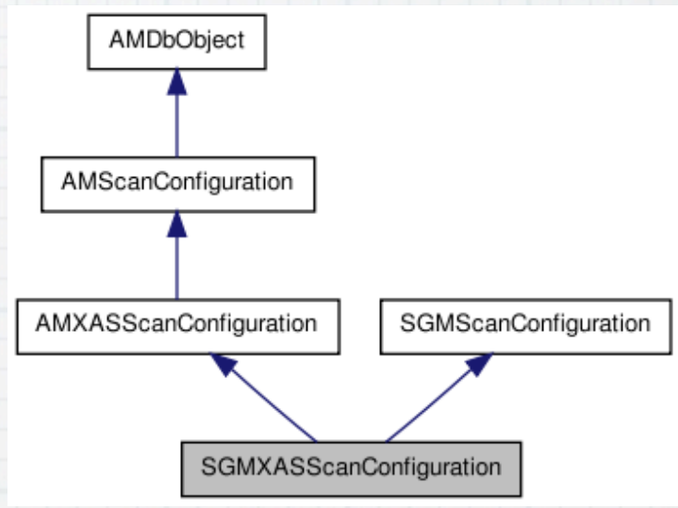
Object Oriented Programming

* Programming paradigm
* Benefits: Inheritance, Polymorphism, and Decoupling
* Framework examples

Inheritance

- Key Benefits:
  - Defining top level APIs
  - **Writing less code**
  - Polymorphism
  - Extending Functionality





- **General behavior** is defined by API/superclass, **specific functionality** is programmed into subclasses
- "Code" examples

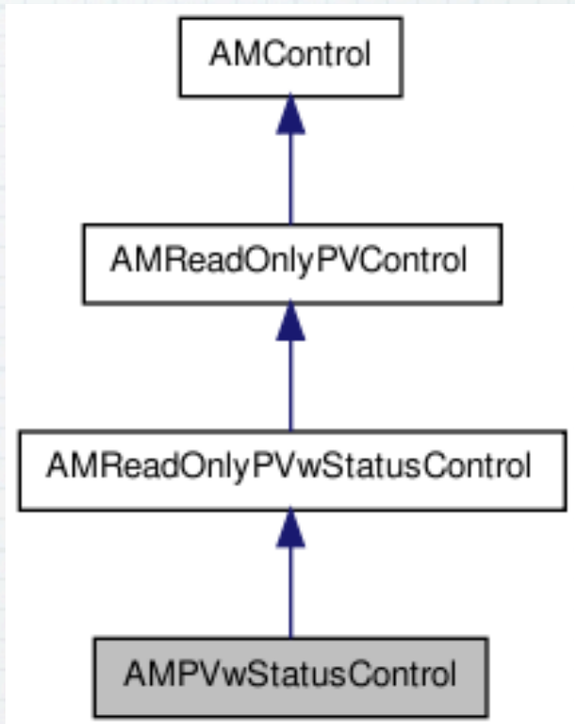
Top Level APIs

* AMControl:

Capabilities

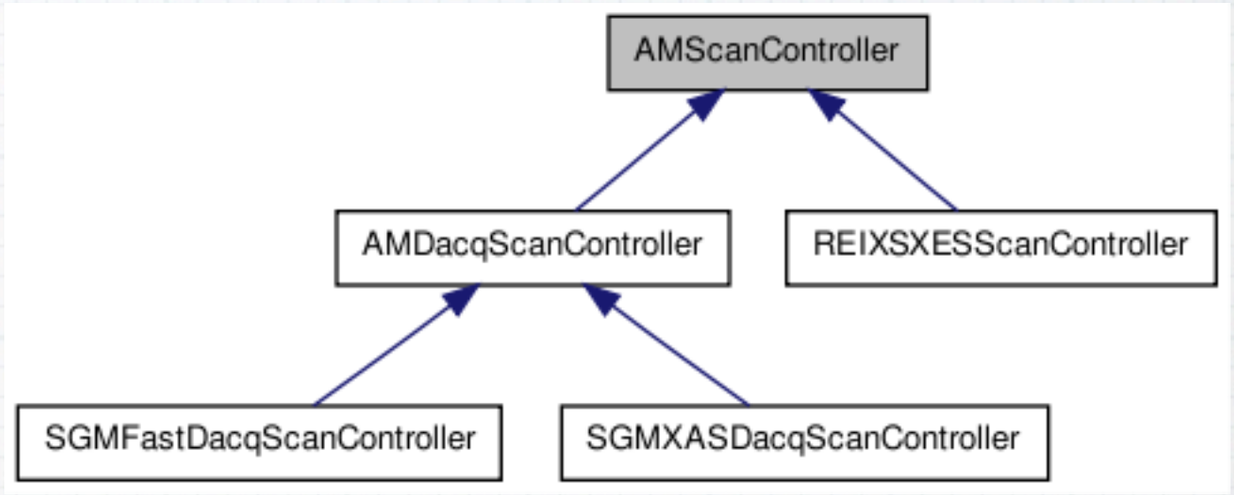
These indicate the current cabilities and status of this control. Unconnected controls can't do anything.

virtual bool	**isConnected** () const Indicates a "fully-functional" control, ready for action.
virtual bool	**canMeasure** () const Indicates that this control *can* currently take measurements.
virtual bool	**shouldMeasure** () const Indicates that this control type *should* (assuming it's connected) be able to measure values.
virtual bool	**canMove** () const Indicates thatthis control *can* (currently) move to setpoints:
virtual bool	**shouldMove** () const Indicates that this control *should* (assuming it's connected) be able to move to setpoints:
virtual bool	**canStop** () const Indicates that this control *can* (currently) issue **stop()** commands while moves are in progress.
virtual bool	**shouldStop** () const Indicates that this control *shoule* (assuming it's connected) be able to issue **stop()** commands while moves are in progress.



Writing Less Code

* (For you, not me)
* AMScanController:

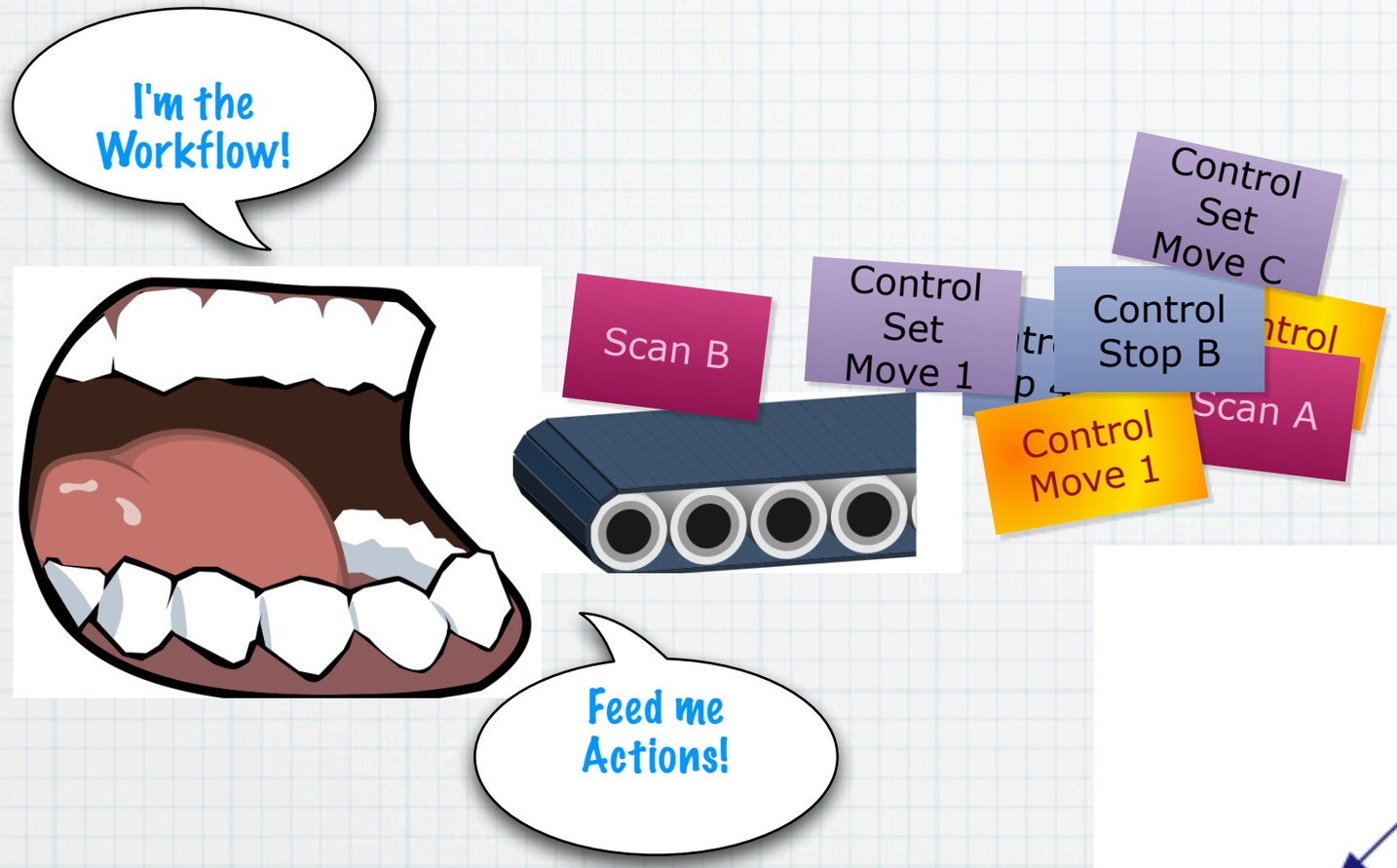


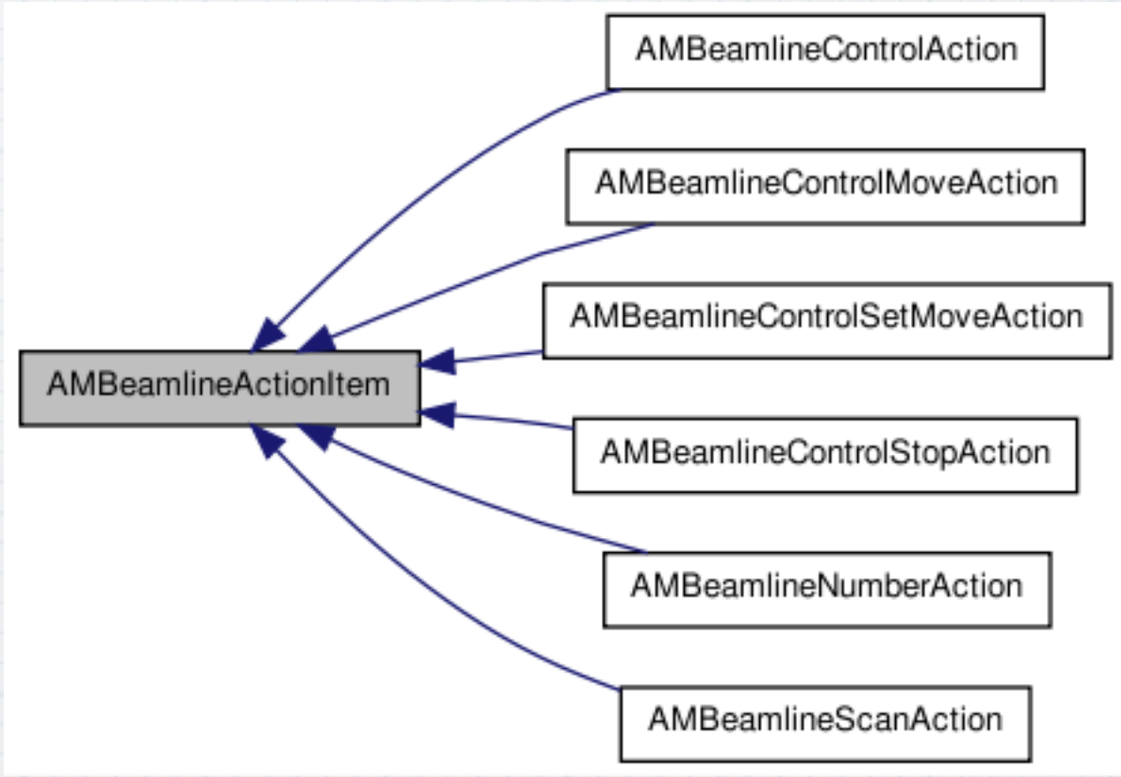
bool	**start** () Start scan running if not currently running or paused. Return value is whether or not we are capable of this transition from the current state (we can only start if we are in the initialized state)

virtual void	**startImplementation** ()=0 Implement to start a scan (ie: transition from Initialized to Running). After the scan is running, call **setStarted()**.

Polymorphism

* Workflow and Actions:





Extending Functionality

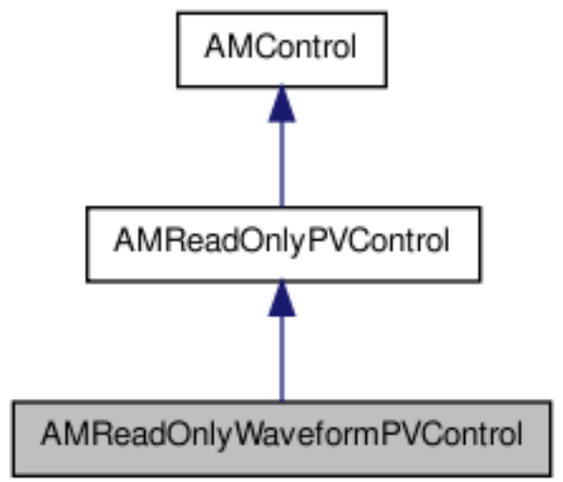
* SDD Waveform:

```
AMReadOnlyWaveformPVControl::AMReadOnlyWaveformPVControl(const QString &name, const QString &readPVname,
                AMReadOnlyPVControl(name, readPVname, parent)
{
        disconnect(readPV_, SIGNAL(valueChanged(double)), this, SIGNAL(valueChanged(double)));
        setBinParameters(lowIndex, highIndex);
        connect(readPV_, SIGNAL(valueChanged()), this, SLOT(onReadPVValueChanged()));
}

double AMReadOnlyWaveformPVControl::value() const{
        return readPV_->binIntegerValues(lowIndex_, highIndex_);
}

void AMReadOnlyWaveformPVControl::setBinParameters(int lowIndex, int highIndex){
        lowIndex_ = lowIndex;
        highIndex_ = highIndex;
}

void AMReadOnlyWaveformPVControl::onReadPVValueChanged(){
        emit valueChanged(value());
}
```



Decoupling

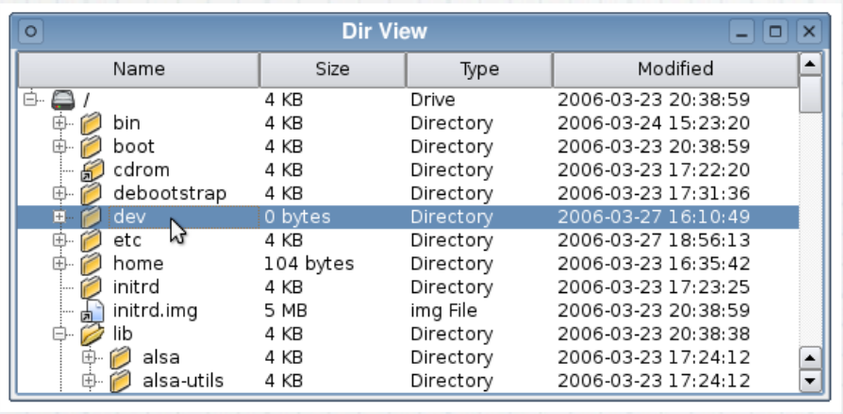
* Key Benefits:
  * Separating interactions
  * Digestible code
  * Reusable objects
* Central feature of **Model/View programming**

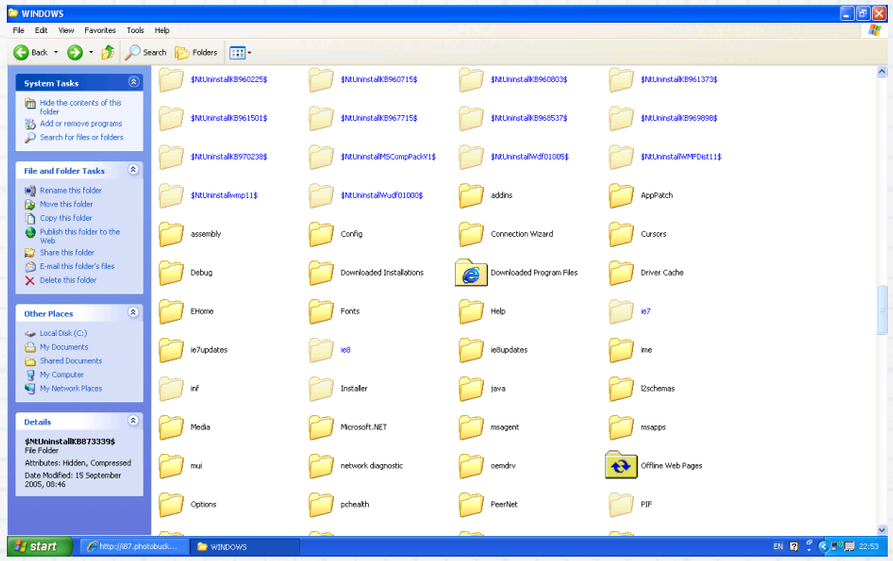
Model/View Programming

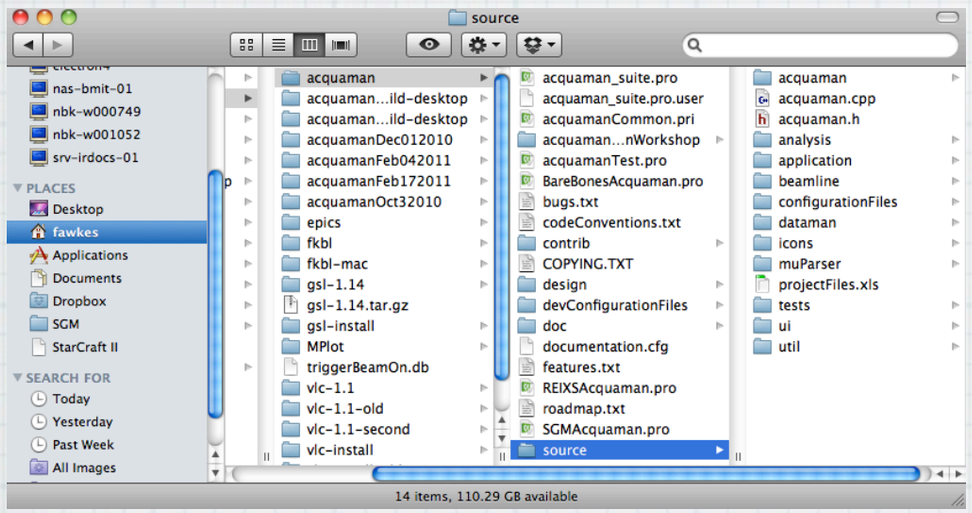
* Generally Model/View/Controller (MVC), **Qt lumps the View and Controller together** (we're doing the same)
* Decouple View (what it looks like) from Model (how it works)
* Examples

General M/V Example

* File Directory:

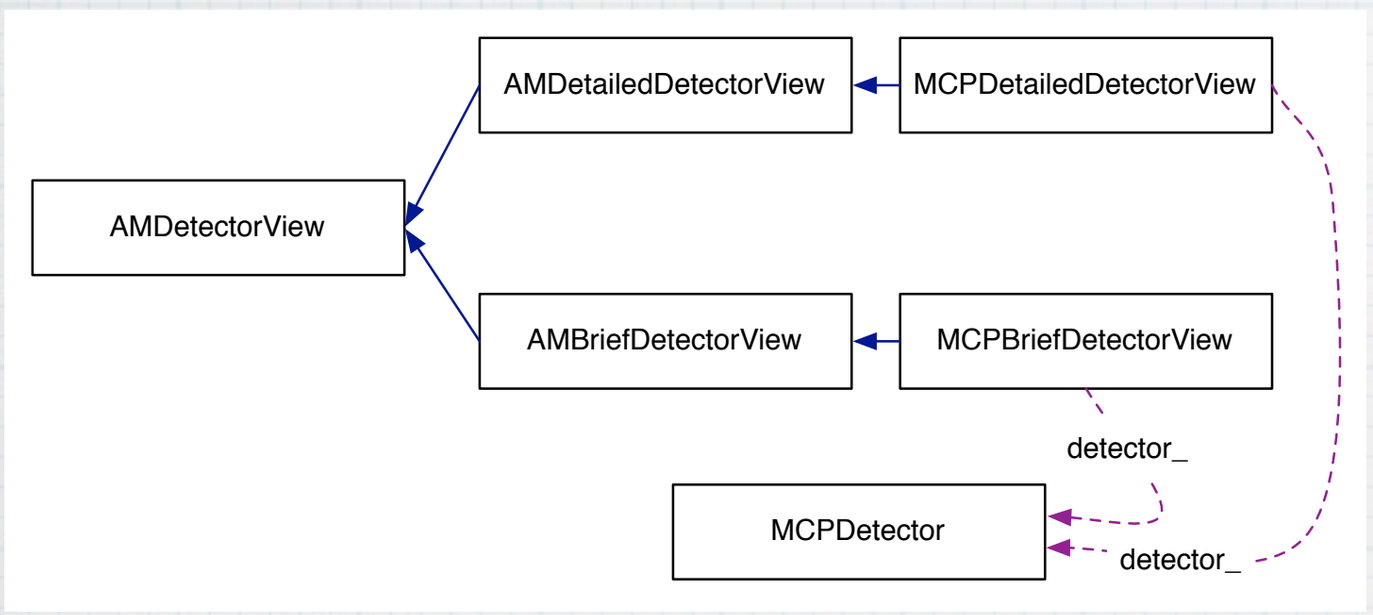


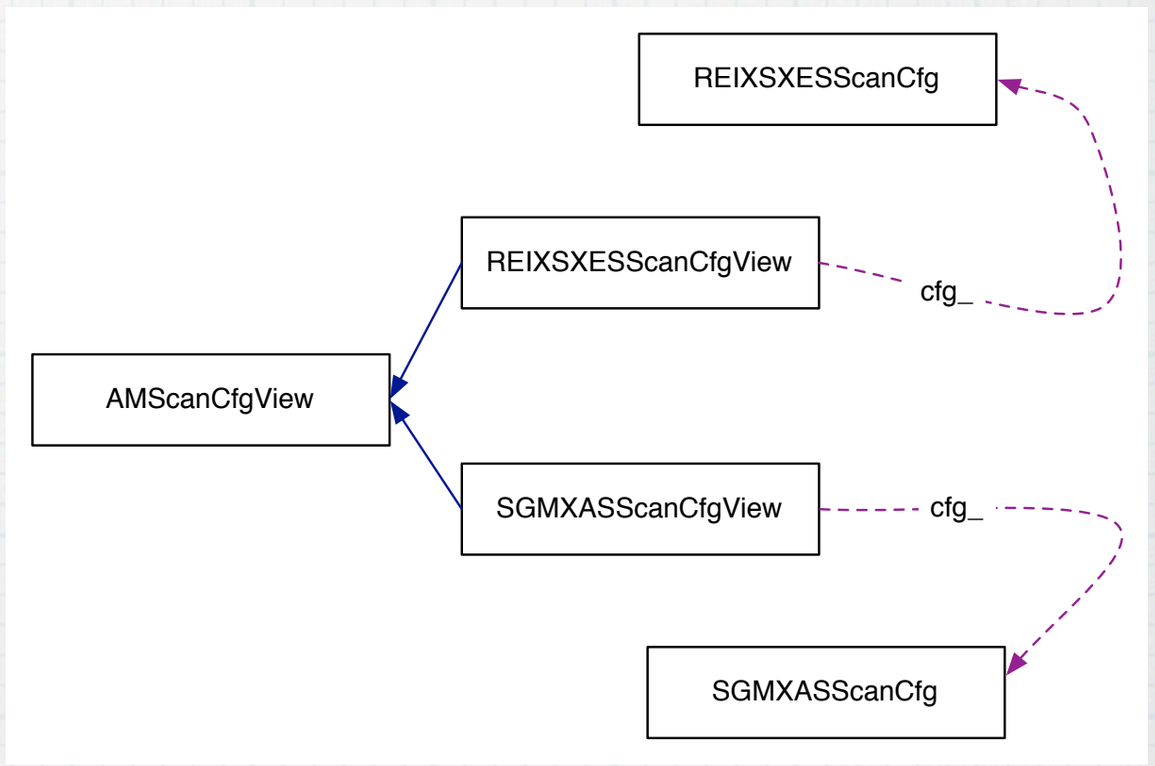




Qt M/V vs Acquaman

* Qt imposes a **very strict** API for models and views (so any model "can" be viewed in their views)
* Acquaman focusses more on the principle
  * Anything that can be "viewed" has a **model** class and a **view** class



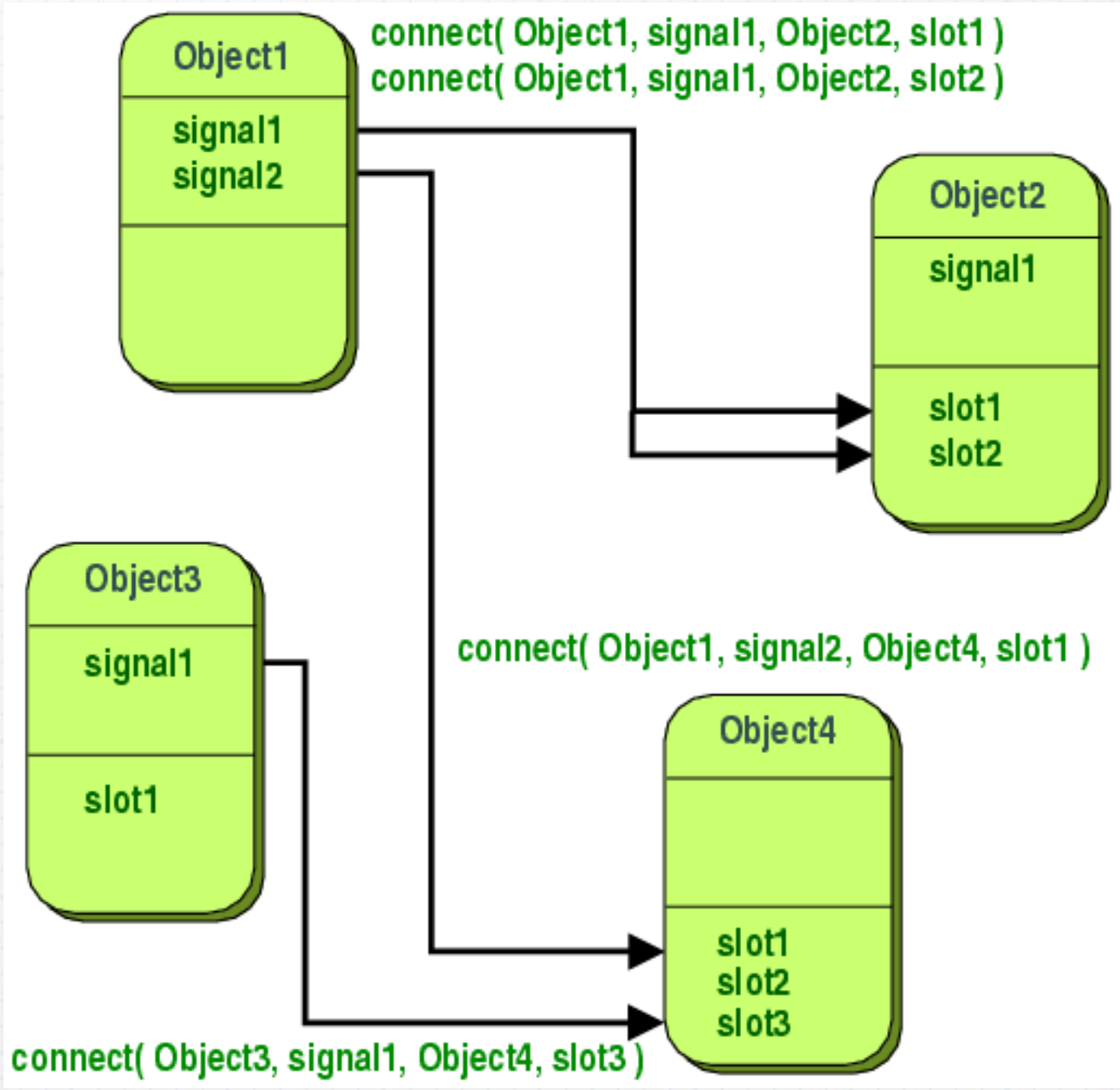


Signals and Slots

* Convenient access to **Event-Driven** Programming Paradigm
  * Acquaman utilizes Signals and Slots to be Event-Driven and manages to be single-threaded (with careful exceptions)
* Benefits:
  * Like callbacks, but type-safe
  * No relationship restrictions (one-to-one, many-to-one, etc.)
  * True information encapsulation
* How it works...

Signals & Slots (Abstractly)

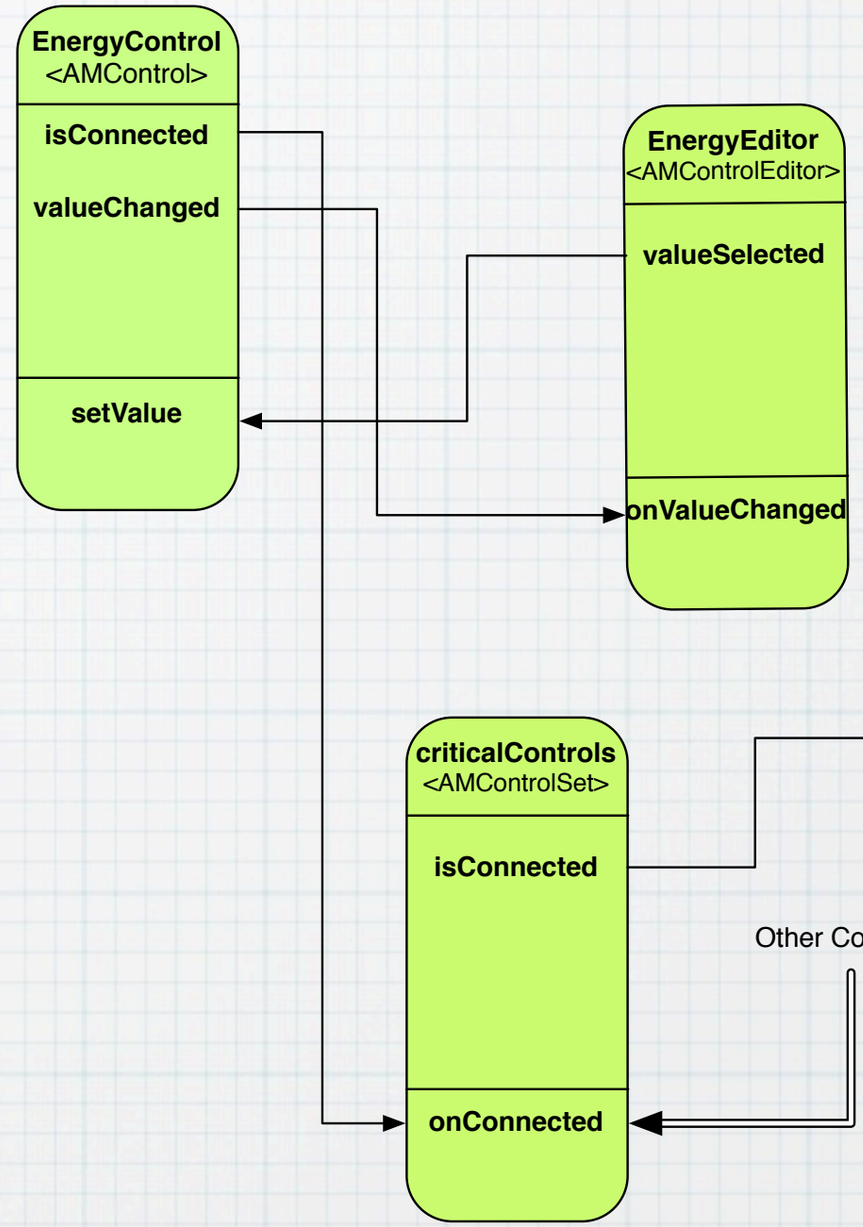
* QObjects (anything that inherits it) can **declare** signals & slots
* Signals can be **"emitted"** when something interesting happens
* Signals can be **"connected"** to slots (and carry payload)



Slots: just member functions, defined under the `slots:` keyword

Signals & Slots (Example)

* We can be event-driven from **EPICS level**: The energy value changed because that **PV changed value**



* We can be event-driven from the **user interface**: User changes energy to some value and hits enter
* Can be done manually:

```
connect(energyControl, valueChanged, EnergyEditor, onValueChanged)
```

* Or automatically: AMControlEditor is constructed with an AMControl

Documentation!

Code Conventions

Source Control

How to get it

Support Systems

Documentation

* Integrated into source code
* (doxygen)
* Online, browse-able
* Updated with every commit:
  * http://beamteam.usask.ca/mark/acquaman (for now)

Documentation

* Currently:
  * a two-sentence "why" description for each class
  * a brief explanation of each function and its parameters.
* Not yet (but we know we need it)
  * A detailed "how" description for each class
  * Code examples

'Acquaman: Data Acquisition and Management for Beamlines': AMControl Class Reference

http://beamteam.usask.ca/mark/acquaman/d4/d9c/class_a_m_control.html#_details

Introduction to Acq... | Aquaman - Televis... | LibXml2 | Doxygen | Doxygen | 'Acquaman: Data A... | source/main.cpp a...

'Acquaman: Data Acquisition and Management for Beamlines' 0.1a

Main Page | Related Pages | Modules | Classes | Files | Search

Class List | Class Index | Class Hierarchy | Class Members

- AMBeamlineActionsListView
- AMBeamlineActionsQueue
- AMBeamlineActionView
- AMBeamlineCameraWidget
- AMBeamlineControlAction
- AMBeamlineControlMoveAction
- AMBeamlineControlMoveActionView
- AMBeamlineControlSetMoveAction
- AMBeamlineControlSetMoveActionView
- AMBeamlineControlStopAction
- AMBeamlineNumberAction
- AMBeamlineParallelActionListModel
- AMBeamlineParallelActionsList
- AMBeamlineParallelActionsListHolder
- AMBeamlineScanAction
- AMBeamlineScanActionView
- AMBiHash< T1, T2 >
- AMBriefDetectorView
- AMCloseItemDelegate
- AMColorControlOptimizationSetView
- AMCompactControlOptimizationSetView
- AMControl
- AMControlButton
- AMControlEditor
- AMControlEditorStyledInputDialog
- AMControlInfo
- AMControlInfoList

bool wasConnected_
Used to detect changes in isConnected()

Detailed Description

An AMControl is an abstract representation of all basic scientific quantities that can be adjusted (controlled) or measured.

Fundamentally, a Control can be "measured" and/or "moved". All controls have the following abilities:

actions:

- void move(double setpoint): set a Control to a requested target value

properties:

- double value(): the readout/value/position at this instant in time
- QString name(): A descriptive (and hopefully unique) name for this Control

- isConnected(): indicates that the control is hooked up and operating correctly; ready for use.
- signal: connected(bool isConnected): emitted for changes in isConnected()

- bool shouldMeasure(): this control should be capable of reporting it's current value, assuming that it's connected.
- bool shouldMove(): this control should be capable of being adjusted, assuming that it's connected.
- bool canMeasure(), canMove(): indicate that, at this instant in time, the control is capable of being measured or moved

- QPair<double, double> range()
- double minimumValue():
- double maximumValue(): the limits of this control's range

- double setpoint(): For Controls that can be moved, this is the most recently requested target (desired value)
- double tolerance(): This "deadband" or "epsilon" represents how close a AMControl must get to it's target value for it to count as inPosition() (or for the move to have succeeded). It represents the level of accuracy the scientist requires for this quantity; ie: the maximum acceptable error between measure() and setpoint(). The default value is CONTROL_TOLERANCE_DONT_CARE -- a very large number. Use setTolerance() to change it.
- bool inPosition(): indicates the AMControl is within tolerance() of it's requested target.

Monitoring Motion

There are two useful but distinct ideas on whether a control is "moving", particularly for distributed control systems like in a synchrotron. Often, beamline parameters can be controlled from a user's software interface, but they can also spontaneously start moving due to external events or other control interfaces.

AMControl

Generated on Tue Mar 15 2011 22:30:55 for 'Acquaman: Data Acquisition and Management for Beamlines' by doxygen 1.7.3

* Good example: AMControl

Source Control

* Collaborative development
* Version control
* Powerful branching and merging
  * ex: Apply a bug fix in the master branch... Merge just that fix into a stable release branch.
* Accessible to non-CLS collaborators

---

* git !

git?

* Open-source version control system
* used by large projects like Qt, WebKit, OpenSUSE, Ruby, Linux (kernel), etc.
* Distributed (each working directory is a complete repository, with history. Don't need a server)
  * Share changes through merging with a "remote" repository
* Focus on fast, easy branching and merging
* Commits: Think about "changes", not "files"
* Crash course: http://gitref.org/

Git basics

* "Clone" a repository to start working:
  `git clone git@github.com:acquaman/acquaman.git`
* Do some work (Make changes to files)
* See what you've done:
  `git status`
* Commit your changes:
  `git commit -a -m "My detailed commit message"`
* Retrieve and merge in other people's changes:
  `git pull`
* Push your changes back to the remote
  `git push`

Source Control

* Remote Repository:
  * Hosted on GitHub
    (public accessibility, no-worry backups, off-site)
    * **git address:**
      `git@github.com:acquaman/acquaman.git`
    * **Web interface**
      https://github.com/acquaman/acquaman
* Secondary remote repository:
  * `git@beamteam.usask.ca/acquaman.git`

Getting Acquaman

* Prerequisites: Development Environment
  * Qt 4 (development libraries and build tools)
  * [optional] Qt Creator (a brilliant IDE)
  * Both available here:
    http://qt.nokia.com/downloads
  * Min version: 4.6,
    but as a rule we keep up-to-date with the latest stable release (4.7.2 now)

Getting Acquaman

* Prerequisites: Development Libraries

Library	Version (min)	Version (max)
EPICS	3.14.9 (oldest tested)	3.14.12 (latest)
GSL (GNU Scientific Library)	whatever	whatever
VLC (Required for video widgets only)	1.1 branch	1.1.8 (latest)
libxml2 (Required for CLS dacqlib)	whatever	whatever
MPlot (plotting library; part of acquaman project)	https://github.com/acquaman/MPlot	

Getting Acquaman: 1

* 1 - Download latest **Acquaman** and **MPlot** (from GitHub website, or clone using git)
  * If you want to use the standard project paths, we usually work in `$HOME/dev` (on Mac), or `$HOME/beamline/programming` (on Linux)

```
cd ~/dev
git clone git@github.com:acquaman/acquaman.git
# (This creates an acquaman folder for you)
git clone git@github.com:acquaman/MPlot.git

# (This creates an MPlot folder for you)
```

Getting Acquaman: 2

* 2 - Get the required libraries
* Shortcuts:
  * Mac: edit and run (or look at and follow) `acquaman/contrib/getAcquamanDependencies_Mac.sh`
  * Linux: most decently-modern systems should be well-equipped already. Might need a newer VLC (*)
  * Windows:

Getting Acquaman: 3

* 3. Create a new Qt project
(in Qt Creator: File > New File or Project...)
  * Include `acquamanCommon.pri` in your project file:

```
include ( acquamanCommon.pri )

TARGET = XYZAcquaman

# Add your UI files here (Qt Creator does automatically when
creating them using File > New File or Project... )
FORMS +=

# Add your header files here (Qt Creator does automatically)
HEADERS +=

# Add your source files here (Qt Creator does automatically)
SOURCES += source/application/XYZ/XYZMain.cpp
```

Acquaman Projects

* The file `acquamanCommon.pri` is a Qt Project "include" file.
  It references all of the acquaman framework code, as well as instructions for linking, include paths, etc.

* Edit it as required if there are special paths to your development libraries

```
# ##################################################################
# QMake project file for acquaman.          January 2010. mark.boots@usask.ca
# Note: Set EPICS_INCLUDE_DIRS, EPICS_LIB_DIR, VLC_*, and GSL_* correctly for platform
# ##################################################################

# Automatically determines a user's home folder
HOME_FOLDER = $$system(echo $HOME)

macx {

    # Where you want to do your acquaman development (as a path from $HOME). You don't need to include leading or trailing slashes.
    DEV_PATH = dev

    # EPICS Dependencies:
    EPICS_INCLUDE_DIRS = $$HOME_FOLDER/$$DEV_PATH/acquaman/contrib/base-3.14.12/include \
        $$HOME_FOLDER/$$DEV_PATH/acquaman/contrib/base-3.14.12/include/os/Darwin
    EPICS_LIB_DIR = $$HOME_FOLDER/$$DEV_PATH/acquaman/contrib/base-3.14.12/lib/darwin-x86

    # MPlot Source
```

Acquaman Projects

* Example ready for you: `acquaman_suite.pro`
* includes as sub-projects:
  * `REIXSAcquaman.pro`
  * `SGMAcquaman.pro`
  * `BareBonesAcquaman.pro`
  * Unit test program

AM Projects: an Example

```
include ( acquamanCommon.pri )

TARGET = REIXSAcquaman

FORMS +=

HEADERS +=     source/application/REIXS/REIXSAppController.h \
     source/acquaman/REIXS/REIXSXESScanConfiguration.h \
     source/acquaman/REIXS/REIXSXESScanController.h \
     source/beamline/REIXS/REIXSBeamline.h \
     source/ui/REIXS/REIXSXESScanConfigurationView.h \
    source/dataman/REIXS/REIXSXESMCPDetectorInfo.h \
    source/acquaman/REIXS/REIXSXESMCPDetector.h \
    source/dataman/REIXS/REIXSXESCalibration.h \
    source/ui/REIXS/REIXSXESMCPDetectorView.h \
    source/ui/REIXS/REIXSXESScanConfigurationDetailedView.h \
    source/ui/REIXS/REIXSXESHexapodControlEditor.h


SOURCES +=     source/application/REIXS/REIXSAppController.cpp \
     source/application/REIXS/REIXSMain.cpp \
     source/acquaman/REIXS/REIXSXESScanConfiguration.cpp \
     source/acquaman/REIXS/REIXSXESScanController.cpp \
     source/beamline/REIXS/REIXSBeamline.cpp \
     source/ui/REIXS/REIXSXESScanConfigurationView.cpp \
    source/dataman/REIXS/REIXSXESMCPDetectorInfo.cpp \
    source/acquaman/REIXS/REIXSXESMCPDetector.cpp \
    source/dataman/REIXS/REIXSXESCalibration.cpp \
    source/ui/REIXS/REIXSXESMCPDetectorView.cpp \
    source/ui/REIXS/REIXSXESScanConfigurationDetailedView.cpp \
    source/ui/REIXS/REIXSXESHexapodControlEditor.cpp
```

* REIXSAcquaman.pro:
* Include acquaman framework
* Add additional (REIXS-specific) code
* Note: nicely organized into REIXS subfolders to keep clean

Structure of a typical AM project ("XYZ")

File(s)	Purpose
`XYZAcquaman.pro`	Qt "Project" file. Specifies all app-specific code files.
`acquamanCommon.pri`	Includes all general Acquaman framework code.
`source/application/XYZ/XYZMain.cpp`	Application entry point: main(). Create your XYZAppController and tell it to `startup()`. [See BareBonesMain.cpp for example]
`source/application/XYZ/XYZAppController.h`	Subclass of AMAppController. Creates and manages all your custom top-level program objects and windows.

Code Conventions

* Readability
* Consistency
* Predictability
* Trivial:
  * Tabs, not spaces, for indenting
  * `useCamelCase()` for variables and `functionNames()`. (No underscores)
  * Member variables have a trailing underscore:

```
protected:
    /// The color of this sheep
    QColor color_;
    /// How many pounds of wool are on it
    double currentWoolWeightPounds_;
```

  * `ALLCAPSFORMACROS / DEFINES`

Code Conventions

* Readability
* Consistency
* Predictability

* Naming:
  * All framework classes (ie: general, common) have an AM prefix (ex: `AMControl`, `AMScan`, `AMDataSource`)
  * Beamline or app-specific code also has a telltale prefix: (`REIXSXESScanController`, `SGMXASScanConfigurationView`, etc.) and goes into its own folder within a module.
  * Yes, we're verbose...

Code Conventions

* Readability * Consistency * Predictability

* Predictable interfaces (following after Qt):

Variable `color_`	`QColor color_;`
Accessor Function `color()`	`QColor color() const;`
Modifier Function `setColor()`	`void setColor(const QColor& newColor);`
Signal `colorChanged()`	`void colorChanged(const QColor& newColor);`

Recap

* Documentation is online
* Get Acquaman using git
  (or from the GitHub website)
* Code conventions for readability,
  consistency, and predictability

Modules

ui

MPlot

acquaman (acquisition)

analysis

beamline

dataman

beamline	Interface to beamline hardware; Organization of controls
dataman	Representation of user data (scans, scan configurations, raw data); access to the user's database
acquaman	How to acquire data and store it (ex: Scan Controllers)
analysis	Standard and custom "filters" for processing data. (binning, peak fitting, 2D-1D summing, etc.)
(MPlot)	Library created for 2D plotting (independent, but used by Acquaman)
ui	All user interface code.
[util]	Miscellaneous utilities (ex: data and time formatting, etc.)
[application]	main() functions and application-wide controller objects

1) beamline

11 beamline

Functionality Overview

* 1) Interface to beamline hardware
  * Implementation-specific AND general (how?)
* 2) Handy organization of controls, so that they can be accessed by programmers

Beamline Core Classes

AMBeamline — Organizes a beamline of controls in a hierarchy.
"Singleton": access from anywhere: `AMBeamline::bl()`

AMControl — Interface to anything you can `measure()` and `move()`

AM____Control — Common implementations of actual controls (*)

AMDetector — Interface for all detectors

AMBeamlineActionItem and kids — Interface for "Beamline Actions"
Implementations can: run scans, move motors, change samples, etc.

AMProcessVariable — Interface to EPICS channel access, with handy signals and slots: `value()`, `setValue()`, `valueChanged()`

Starting simple: How to change a PV?

AMProcessVariable

```
// Let's assume we have one already:
AMProcessVariable* myPV = MyBeamline::bl()->myPV1();


// Or create and connect one:
AMProcessVariable* myPV =
     new AMProcessVariable("PCT1402-01:mA:fbk", true);
```

v Channel Access Name

start monitoring right now please ^

Starting simple: How to change a PV?

```
// Is it connected?
myPV->isConnected();  // true  (After some time, hopefully)

  // Or check permissions and connection status:
  myPV->canRead();    // true
  myPV->canWrite();  // also true (interesting, for the ring
current)

  // Other handy tricks:
  myPV->units();      // "mA"
```

Starting simple: How to change a PV?

```
myPV->setValue(3.0);
myPV->setValue(QString("3.0"));
myPV->setValue(3);                                  // Can use as slots too!

myPV->lastValue(); // 3.0 (most recent monitored value)
                   // (If monitoring is on)

myPV->getInt();     // 3
myPV->getDouble(); // 3.0    // Converted for you, if not the native type
myPV->getString(); // "3.0"

connect(myPV, SIGNAL(valueChanged(double)), someObject,
SLOT(handleNewValue(double)));    // someObject will now get this signal
                                  //  on every PV monitor event
```

Starting simple: How to change a PV?

```
// Support array PVs automatically:

myPV->count(); // 1 (for this one)... more for others

  // If there were more:
  myPV->lastValue(index); // value at index
  myPV->lastFloatingPointValues();  // the whole array, as a
QList<double>

  myPV->setValues(doubleArray, arraySize);
  myPV->setValues(intArray, arraySize);
  myPV->setValues(stringList);
```

Abstraction: AMControl

* EPICS Channel Access only one way (but common) to access hardware.
* What are the essential characteristics of a "controllable" object?

value() ?

setpoint() ?

tolerance() ?

isMoving() ?

// My move? Someone else moved it?

move()

stop()

// some can...

Abstraction: AMControl

* Even at a general level:
  * Uniqueness... Not all controls can do the same thing...

canMeasure() ?

canMove() ?

canStop() ?

// for programmers:

name() ?

// for humans:

description() ?

units() ?

// Limits!

minimumValue() ?

maximumValue() ?

Abstraction: AMControl

* When do you want to be notified?
* SIGNALS:

connected()

valueChanged()

// any move:

movingChanged()

// the move you asked for:

moveStarted()

moveSucceeded()

moveFailed(why)

AMControl: Implementations

AMReadOnlyPVControl	Just a feedback value (single PV)
AMPVControl	A feedback PV and a setpoint PV [Optional: Stop PV]
AMReadOnlyPVwStatusControl	A feedback PV and a status PV (don't need to guess at moving anymore)
AMPVwStatusControl // Motors!	Feedback, setpoint, and moving status PVs [Optional: Stop PV]

AMControl: Implementations

* Different ways to implement the SAME interface
* Upper layers:
  can just deal with "**AMControls**", and forget about the details
* Other control systems (TINE, TANGO, etc.) or non-EPICS hardware: just create own AMControl implementations and roll...

AMControl: UI Views

AMControlEditor, AMBasicControlEditor

click:


```
AMPVControl* myControl =
        new AMPVwStatusControl("hexapodX",                  // name
                               "HXPD1610-401:X:sp",         // feedback PV
                               "HXPD1610-401:X",            // setpoint PV
                               "HXPD1610-401:moving",       // Moving status
                               QString(),                   // No stop PV
                               this,                        // parent object
                               0.1);                        // tolerance

AMBasicControlEditor* myEditor =
        new AMBasicControlEditor(myControl, this);
```

AMBeamline

* Used to: group and access controls
* Singleton object (only one beamline)
* Can access the object from anywhere in your app:
  * AMBeamline::bl() or XYZBeamline::bl()
* Meant to be subclassed
* Big or small as you need it

```
class SGMBeamline : public AMBeamline
{
     Q_OBJECT

public:
     enum sgmGrating {lowGrating=0, mediumGrating=1, highGrating=2};
     QString sgmGratingName(SGMBeamline::sgmGrating grating) const;
     enum sgmHarmonic {firstHarmonic=1, thirdHarmonic=3};
     QString sgmHarmonicName(SGMBeamline::sgmHarmonic harmonic) const;

     static SGMBeamline* sgm();          // singleton-class accessor

     virtual ~SGMBeamline();

     bool isConnected() const {
          return criticalControlsSet_->isConnected();
     }

     QStringList unconnectedCriticals() const;

     bool detectorConnectedByName(QString name);

     QString beamlineWarnings();

     QString pvName(const QString &amName) const { return amNames2pvNames_.valueF(amName);}
     QString amName(const QString &pvName) const { return amNames2pvNames_.valueR(pvName);}
```

// Singleton Access: SGMBeamline::sgm()

```
  AMControl* ringCurrent() const { return ringCurrent_; }
  AMControl* energy() const { return energy_;}
  AMControl* energySpacingParam() const { return energySpacingParam_;}
  AMControl* energyC1Param() const { return energyC1Param_;}
  AMControl* energyC2Param() const { return energyC2Param_;}
  AMControl* energySParam() const { return energySParam_;}
  AMControl* energyThetaParam() const { return energyThetaParam_;}
  AMControl* exitSlitGap() const { return exitSlitGap_;}
  AMControl* entranceSlitGap() const { return entranceSlitGap_;}
  AMControl* mono() const { return energy_->childControlAt(0);}
  AMControl* undulator() const { return energy_->childControlAt(1);}
  AMControl* exitSlit() const { return energy_->childControlAt(2);}
  AMControl* m4() const { return m4_;}
  AMControl* grating() const { return grating_;}
  AMControl* harmonic() const { return harmonic_;}
  AMControl* undulatorTracking() const { return undulatorTracking_;}
  AMControl* monoTracking() const { return monoTracking_;}
  AMControl* exitSlitTracking() const { return exitSlitTracking_;}
```

// Lots of Controls

```
  AMDetector* teyDetector() const { return teyDetector_;}
  AMDetector* tfyDetector() const { return tfyDetector_;}
  AMDetector* pgtDetector() const { return pgtDetector_;}
  AMDetector* i0Detector() const { return i0Detector_;}
  AMDetector* eVFbkDetector() const { return eVFbkDetector_;}
  AMDetector* photodiodeDetector() const { return photodiodeDetector_;}
  AMDetector* encoderUpDetector() const { return encoderUpDetector_;}
```

// Some Detectors

beamline: other classes

AMControlSet	A group of controls, accessible by index or by name
AMControlInfo	The frozen state (value, min, max, etc.) of a control

Applying beamline

subclass AMBeamline:
XYZBeamline

// long-lived objects

in XYZBeamline constructor,
create your control objects

Make accessor functions to
return (pointers to) controls

[optional]

Make AMControlSets for useful
groups of controls


DATAMAN
ELECTRONIC LEARNING AID
★ Basic math drills with design-your-own practice sessions.
★ Fun math games plus built-in timer & scorekeeper.
★ Scoreboard Spectacular to reward you when you're right.
TEXAS INSTRUMENTS

2) Dataman

dataman Core Classes

AMDatabase — Low-level access to a database (the user's database, public database, whatever). Make tables, rows, queries, etc.

AMDbObject — Base class for all "persistent" objects. Inherit AMDbObject so that you can automatically save yourself to the database (`storeToDb()`), or restore yourself (`loadFromDb()`)

AMScan — AMDbObject that represents a scan. Container for a raw data store, some metadata, raw and filtered data sources.

AMRun / **AMExperiment** — All scans belong to a (single) run. However, users can organize their scans into experiments however they want.

AMUser — We might as well have an AMDbObject for them too! (Preferences, settings, etc.)

dataman Core Classes

AMDataSource

Interface for exposing multi-dimensional data, so that it can be plotted, analyzed, etc.

AMnDIndex

Anywhere we need to specify index into multi-dim data.
`AMnDIndex()`, `AMnDIndex(3)`, `AMnDIndex(2,7)`, etc.

point 1D 2D ...

AMDataStore

API for storing raw (multi-dim) scan data. Implemented by:
`AMInMemoryDataStore`, ~~`AMHDF5MemoryStore`~~ (not yet)

AMAxisInfo

Describes an axis (size, units, etc.) in a dataset.
`QList<AMAxisInfo>` can describe multiple axes.

AMNumber

Doubles and ints don't let us return a "No Data Here" or a "Something's Wrong" value. AMNumber can hold a double or an int, as well as a state.

AMDatabase

low-level DB access

* "Doubleton" class:
  returns user's private / public, databases:

```
static AMDatabase* userdb();
static AMDatabase* publicdb();
```

* Saving/Updating rows:

```
int insertOrUpdate(int id, const QString& table, const QStringList& colNames,
const QVariantList& values);
bool update(int id, const QString& table, const QString& column, const QVariant&
value);
```

* Retrieving Rows or just Values:

```
QVariantList retrieve(int id, const QString& table, const QStringList& colNames)
const;
QVariant retrieve(int id, const QString& table, const QString& colName)
const;
```

AMDatabase

low-level DB access

* Adding tables, columns, indexes:

```
bool ensureTable(const QString& tableName, const QStringList& columnNames, const
QStringList& columnTypes, bool reuseDeletedIds = true);

bool ensureColumn(const QString& tableName, const QString& columnName, const
QString& columnType = "TEXT");

bool createIndex(const QString& tableName, const QString& columnNames);
```

* Start a raw sql query (using the QSQLQuery object)

```
QSqlQuery query() { return QSqlQuery(qdb()); }
```

* Scared?

* Good, don't use it...

AMDbObject

Easy persistent database storage for your objects

* Have an AMDbObject?
  * Store its current state forever:

```
AMScan s;
// Get it full and interesting...
// ...
int databaseId = s.storeToDb(AMDatabase::userdb());
```

* Restore it back
(different instance, same instance, whatever)

```
AMScan s2;
s2.loadFromDb(AMDatabase::userdb(), databaseId);
```

AMDbObject

Easy persistent database storage for your objects

* Making your own AMDbObject:
  * 1) Inherit AMDbObject
  * 2) Use the Q_OBJECT macro
  * 3) Define **properties** for the values you want to be persistent
  * 4) Register your class before attempting to `storeToDb()` or `loadFromDb()`

AMDbObject

Easy persistent database storage for your objects

```
class XYZThingie : public AMDbObject {
   Q_OBJECT

   Q_PROPERTY(double springConstant READ springConstant WRITE setSpringConstant)

public:
   XYZThingie(QObject* parent = 0);
   double springConstant() const { return springConstant_; }

public slots:
   void setSpringConstant(double newSC) {
      springConstant_ = newSC;
   }

protected:
   double springConstant_;
};
```

```
AMDbObjectSupport::registerClass<XYZThingie>();
```

AMDbObject

Easy persistent database storage for your objects

```
class XYZThingie : public AMDbObject {
   Q_OBJECT

   Q_PROPERTY(double springConstant READ springConstant WRITE setSpringConstant)

public:
   XYZThingie(QObject* parent = 0);
   double springConstant() const { return springConstant_; }

public slots:
   void setSpringConstant(double newSC) {
      springConstant_ = newSC;
   }

protected:
   double springConstant_;
};
```

```
AMDbObjectSupport::registerClass<XYZThingie>();
```

AMDbObject

Easy persistent database storage for your objects

```
class XYZThingie : public AMDbObject {
   Q_OBJECT

   Q_PROPERTY(double springConstant READ springConstant WRITE setSpringConstant)

public:
   XYZThingie(QObject* parent = 0);
   double springConstant() const { return springConstant_; }

public slots:
   void setSpringConstant(double newSC) {
      springConstant_ = newSC;
   }

protected:
   double springConstant_;
};
```

1

2

```
AMDbObjectSupport::registerClass<XYZThingie>();
```

AMDbObject

Easy persistent database storage for your objects

```
class XYZThingie : public AMDbObject {
    Q_OBJECT

    Q_PROPERTY(double springConstant READ springConstant WRITE setSpringConstant)

public:
    XYZThingie(QObject* parent = 0);
    double springConstant() const { return springConstant_; }

public slots:
    void setSpringConstant(double newSC) {
        springConstant_ = newSC;
    }

protected:
    double springConstant_;
};
```

1

2

3

```
AMDbObjectSupport::registerClass<XYZThingie>();
```

AMDbObject

Easy persistent database storage for your objects

```
class XYZThingie : public AMDbObject {
    Q_OBJECT

    Q_PROPERTY(double springConstant READ springConstant WRITE setSpringConstant)

public:
    XYZThingie(QObject* parent = 0);
    double springConstant() const { return springConstant_; }

public slots:
    void setSpringConstant(double newSC) {
        springConstant_ = newSC;
    }

protected:
    double springConstant_;
};
```

1

2

3

```
AMDbObjectSupport::registerClass<XYZThingie>();
```

4 (at runtime)

AMDbObject

Easy persistent database storage for your objects

getter function

Type

```
Q_PROPERTY(double springConstant READ springConstant WRITE setSpringConstant)
```

Property name
(DB column name)

setter function

AMDbObject

Easy persistent database storage for your objects

* Supported types (of stored variables)
  * `int`, `double`, `QString`, `QStringList`, `QByteArray`,
  * `AMIntList`, `AMDoubleList`, `AMnDIndex`
  * `QDateTime`
  * everything else:
    attempted to convert to string and stored as text

AMDbObject

Easy persistent database storage for your objects

* Except...

* `AMDbObject*`
  or
  `AMDbObjectList` (aka `QList<AMDbObject*>`)

* Store composite objects! (Child objects)

```
Q_PROPERTY(AMDbObject* scanInitialConditions READ scanInitialConditions WRITE dbLoa
Q_PROPERTY(AMDbObjectList rawDataSources READ dbReadRawDataSources WRITE dbLoadRawD
```

* Special rules for getter and setter functions...
  see documentation for `AMDbObject`.

AMDbObject

Easy persistent database storage for your objects

* Extra options: Q_CLASSINFO("propertyName", "...")
  * create index on columns
  * share a table with another class. (Otherwise each class gets its own)
  * Provide a description of the type
  * Tell it not to re-use row ids after deleting old objects
  * specify a version number (for db upgrades)

AMDbObject

Easy persistent database storage for your objects

* Extra options: Q_CLASSINFO("propertyName", "...")
  * create index on columns
  * share a table with another class. (Otherwise each class gets its own)
  * Provide a description of the type
  * Tell it not to re-use row ids after deleting old objects
  * specify a version number (for db upgrades)

See Documentation

AMScan

One powerful AMDbObject

- What/Why?
  - Provide **metadata** for user scans
  - Act as container/way of accessing:
    - Scan's raw (RAW) data: `AMDataStore: rawData();`
    - Scan's raw data, nicely exposed for analyzing or plotting: `AMRawDataSource: rawDataSources();`
    - Processed data (using analysis module): `AMAnalysisBlock: analyzedDataSources();`
    - The user's scan setup (unique for different kinds of scans): `AMScanConfiguration: scanConfiguration();`
    - The state of the beamline at the start of the scan: `AMControlInfoList: scanInitialConditions();`
  - Load raw data from file: `loadData()` [subclasses: options for file formats]

AMScan

One powerful AMDbObject

- Scan MetaData
  - **name** (user-given), **number** (user-given)
  - **dateTime**
  - **runId** (Which run did I take this on?)
  - **sampleId** (Connection to `AMSample` Db objects)
  - **notes** (Free-form text for user notebook)
  - **fileFormat**, **filePath**, **additionalFilePaths** (for loading raw data)
  - **scanInitialConditions**: state of beamline at start of scan

AMDataSource

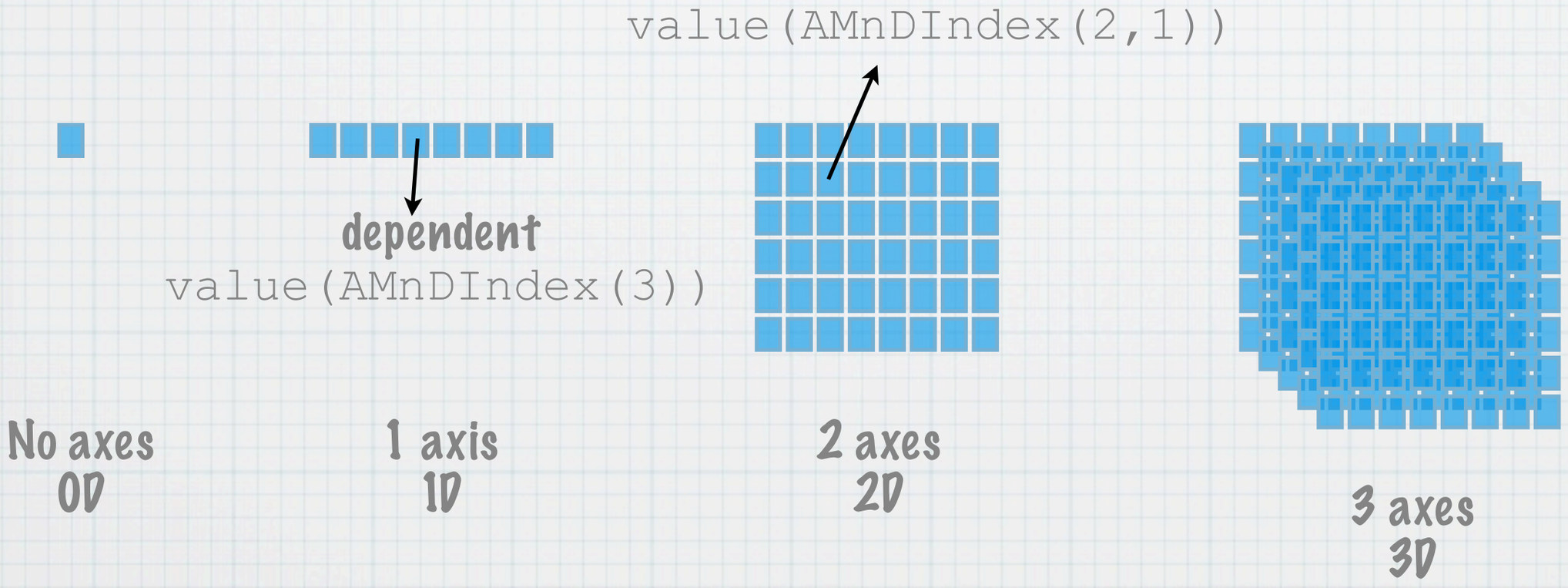
Generic interface to multi-dimensional data

* Anything that implements the `AMDataSource` interface can be understood / analyzed / plotted.
  * Dimensionality, Size, and range of the data:
    `QList<AMAxisInfo> axes();`
  * Current state of the data (valid? processing? static?):
    `int state();`
  * Data (ie: dependent) values:
    `AMNumber value(AMnDIndex i);`
  * Axis (ie: independent) values:
    `AMNumber axisValue(int axisId, int index);`
  * Units, description, identifying name, etc.

AMDataSource

Generic interface to multi-dimensional data

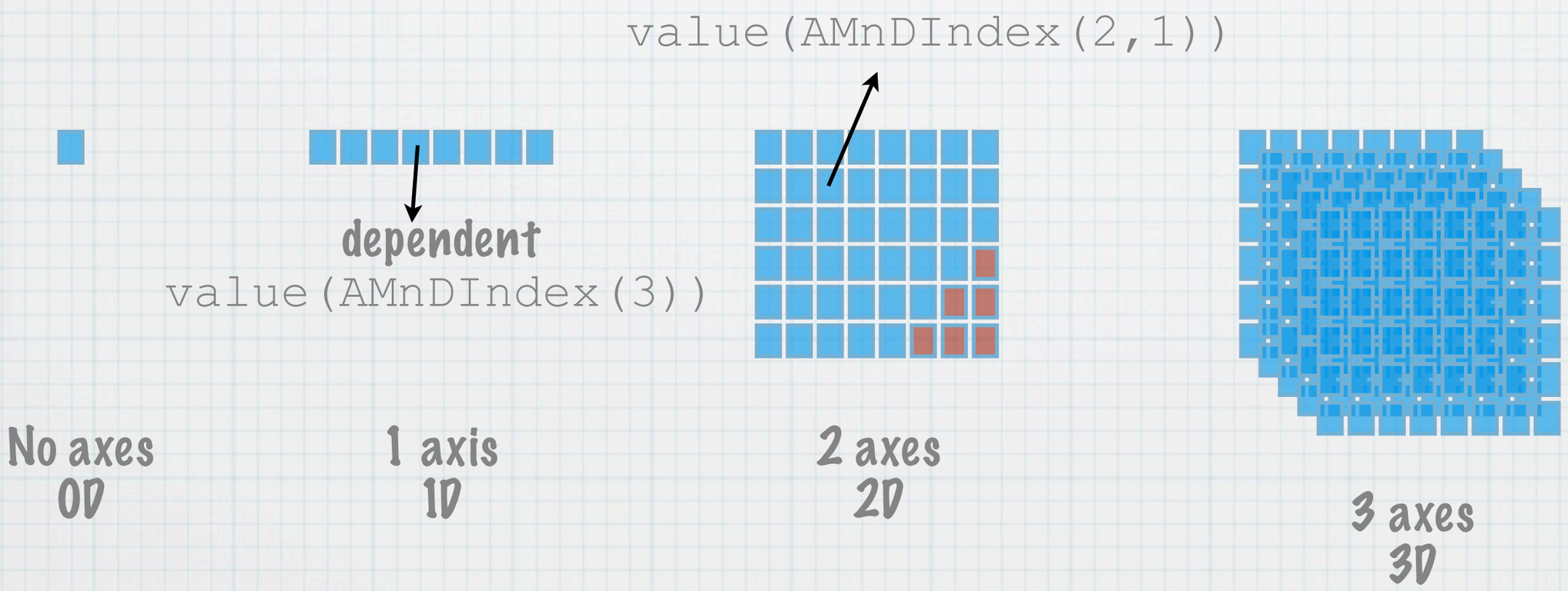
* NOTE: assumes a regular n-dimensional space



AMDataSource

Generic interface to multi-dimensional data

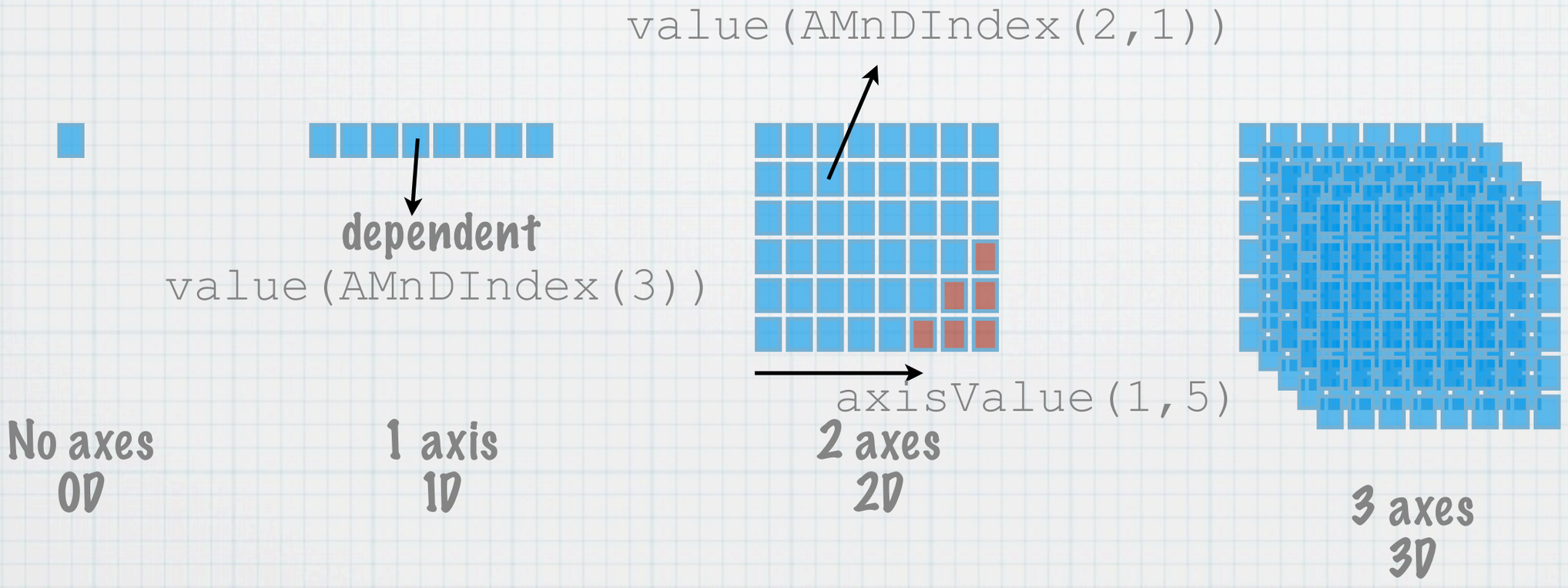
* NOTE: assumes a regular n-dimensional space
* Can return AMNumber::Null for missing values



AMDataSource

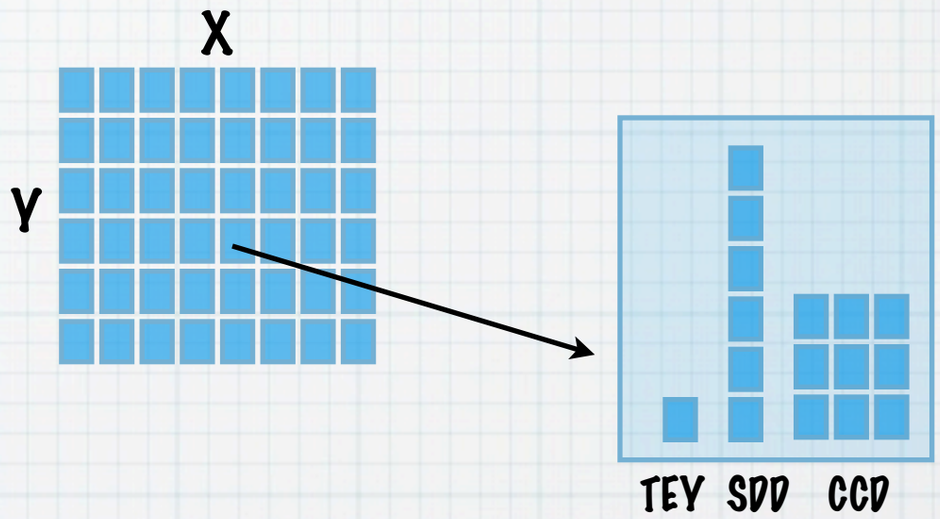
Generic interface to multi-dimensional data

* NOTE: assumes a regular n-dimensional space
* Can return AMNumber::Null for missing values



AMDataStore

* `AMDataSource` is for exposing data... `AMDataStore` is for storing it.
* Problem: Could be a weird shape.
  * Imagine:
    * 2D microprobe scan,
    * using constant incident energy
    * while measuring with a 0D detector (TEY, fluorescence detector), a 1D detector (silicon drift detector), and a 2D detector (CCD or XES image)

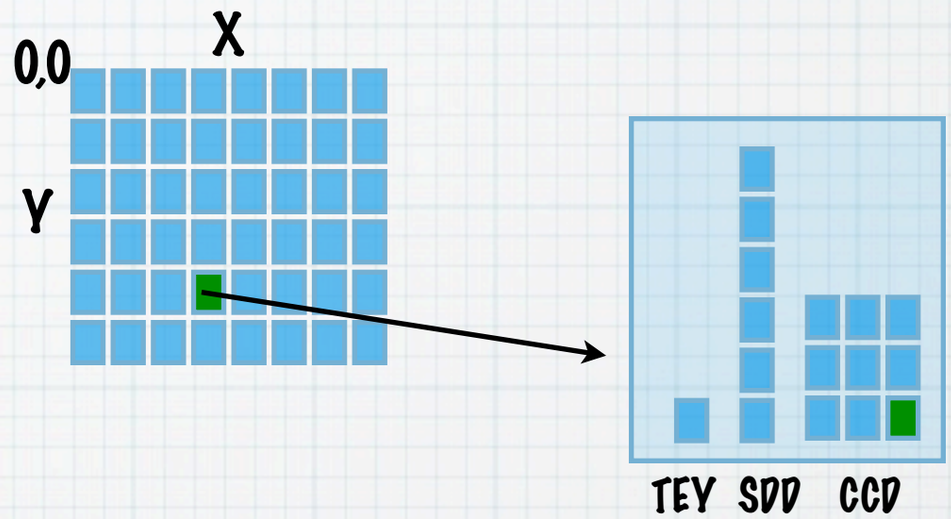


* Large datasets
* Notification when data changes

AMDataStore

- Solve with:
  - Three spaces:
    - **n-dimensional Scan space:**
      - [what you "iterated" over: varied **n** parameters and measured something at each point]
    - **set of detectors (1-dimensional)**
    - **(for each detector) m-dimensional detector space**
  - Different implementations for optimization and performance
    - `AMInMemoryDataStore`
    - `AMHDF5DataStore` **(disk-based with caching)** *//todo*

AMDataStore



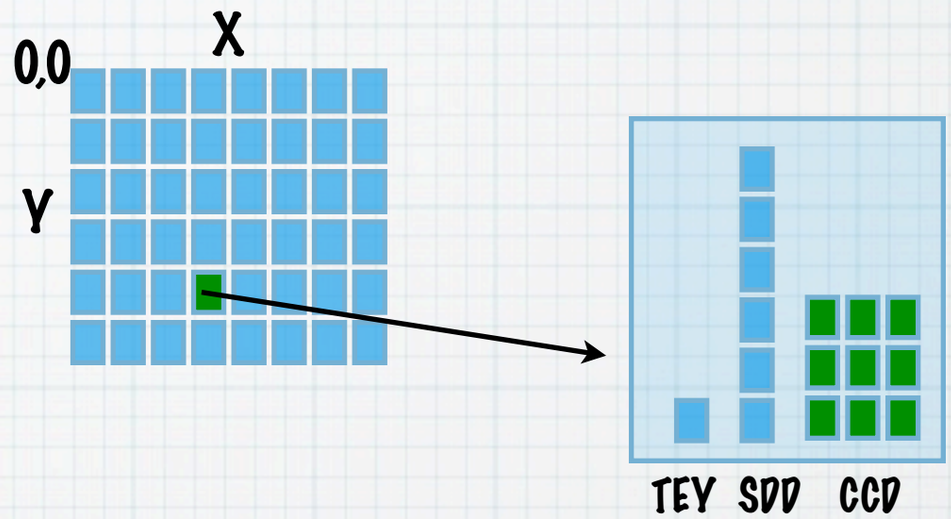
* Back to example:
  * 2D microprobe scan,
  * using constant incident energy
  * while measuring with a 0D detector (TEY, fluorescence detector), a 1D detector (silicon drift detector), and a 2D detector (CCD or XES image)

Scan: X=3 Y=4 Detector: CCD (2) Pixel position: Px = 2, Py = 2

```
/// Set the value of a measurement, at a specific scan point
   virtual bool setValue(const AMnDIndex& scanIndex,
                         int measurementId,
                         const AMnDIndex& measurementIndex,
                         const AMNumber& newValue)
```

AMnDIndex(3, 4)
2
AMnDIndex(2, 2)
37

AMDataStore



* Back to example:
  * 2D microprobe scan,
  * using constant incident energy
  * while measuring with a 0D detector (TEY, fluorescence detector), a 1D detector (silicon drift detector), and a 2D detector (CCD or XES image)

Scan: X=3 Y=4 Detector: CCD (2) Pixel values: [have whole image at once]

```
/// Performance optimization for setValue(): this allows multi-dimensional
measurements to be set in a single setValue call.
   virtual bool setValue(const AMnDIndex &scanIndex,
                         int measurementId,
                         const int* inputData,
                         int numArrayElements)
```

AMnDIndex(3, 4)
2
image
9

AMDataStore: Using

- **Configuring an AMDataStore:**
  - Create scan-space "scan axes":

    addScanAxis(const AMAxisInfo& axisDetails)
  - Add detectors:

    addMeasurement(const AMMeasurementInfo& measurementDetails)
- **Filling an AMDataStore:**
  - Add "rows" (ie: extend axes as you go):

    beginInsertRows(int axisId, int numRows = 1, int atRowIndex = -1)
  - Insert data values: (or use bulk `setValue()` function)

    setValue(const AMnDIndex& scanIndex, int measurementId, const AMnDIndex& measurementIndex, const AMNumber& newValue)
  - Finish inserting and notify automatically that there's new data:

    void endInsertRows();

back to AMDataSource

Implementations: Almost anything

* **Done for you: AMRawDataSource**
  Easily expose one "detector" or "measurement" out of a data store.
  * Dimensionality: Scan space first, then added dimensions for the measurement space
  * Ex: 2D micromap scan with 1D SDD detector = 3D
    Axis 0: X motion
    Axis 1: Y motion
    Axis 2: SDD detector axis (fluorescence energy)

```
                                                          // AMDataStore
scan()->addRawDataSource(new AMRawDataSource(scan()->rawData(), 0));
                                                                // Detector # 0
```

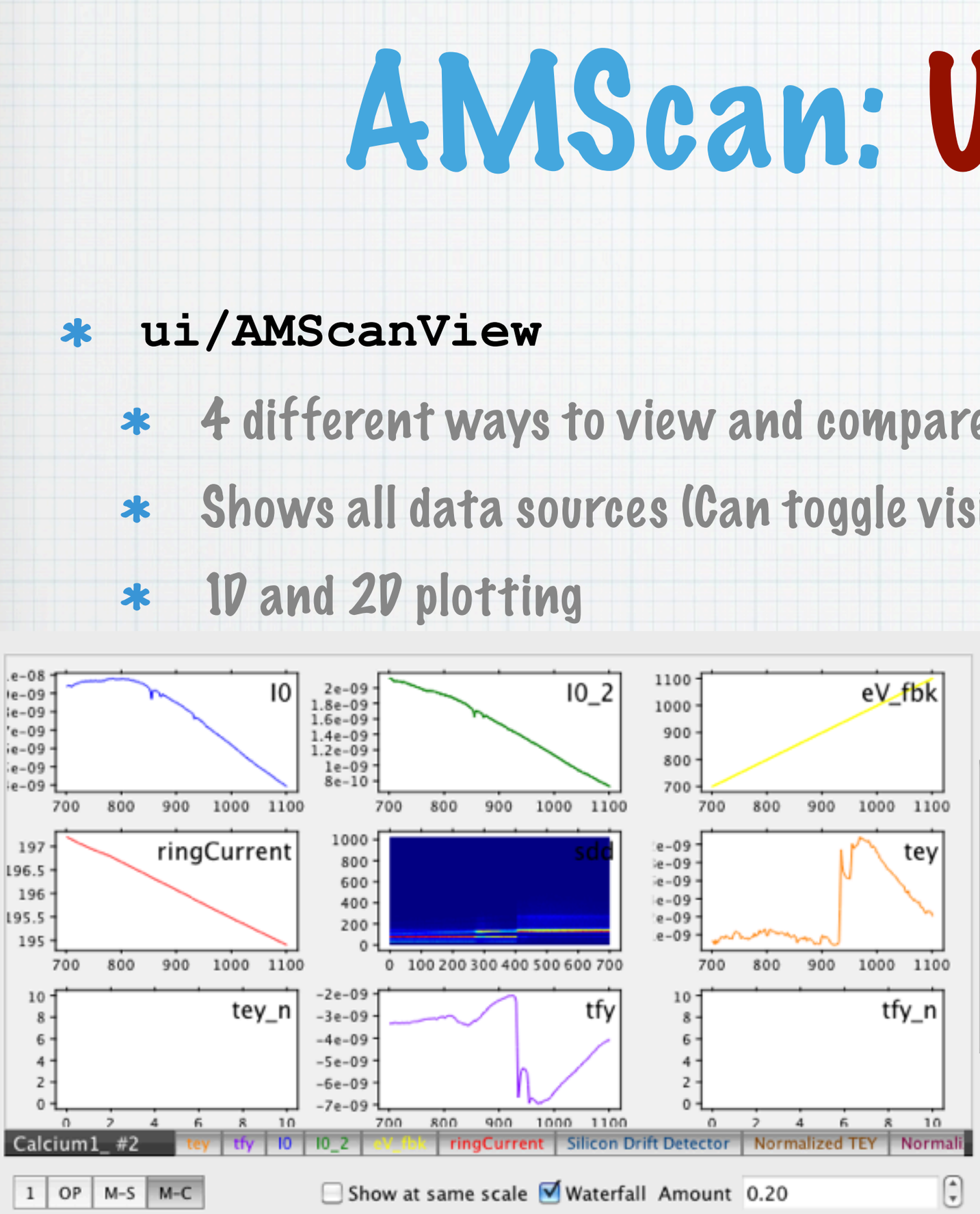
back to AMDataSource

Implementations: Almost anything

- All `AMAnalysisBlocks`
  - See analysis module
  - Basically: analysis blocks are "filters" that eat sets of `AMDataSources`.
    - They themselves are also `AMDataSources`
      - Creates a "chain" of filters processing data.
      - Can plot / visualize at any stage. `AMScan` shows you all your `AMDataSources`:
        `dataSourceCount()` and `dataSourceAt(index);`
- Anything you build, that you want to use for:
  - Plotting
  - Input to an analysis filter

AMScan: UI Views

- **`ui/AMScanView`**
  - 4 different ways to view and compare the data sources in a group of scans
  - Shows all data sources (Can toggle visible or not)
  - 1D and 2D plotting

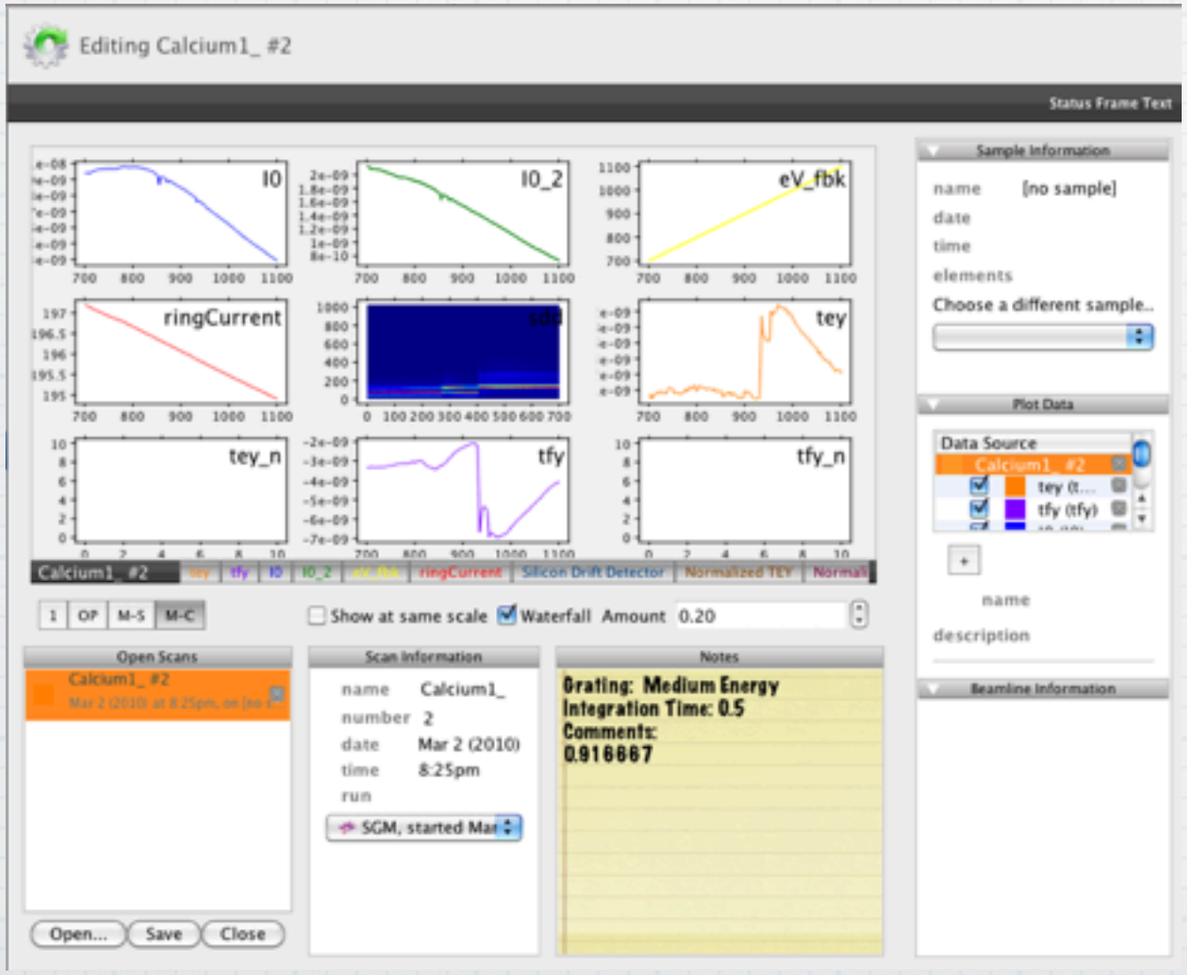



```
AMScanView* view = new AMScanView();
AMScan* scan1 = getFromSomewhere();
AMScan* scan2 = getFromSomewhere();

view->addScan(scan1);
view->addScan(scan2);
```

AMScan: UI Views

* `ui/AMGenericScanEditor`
  * An AMScanView and more...
  * View and modify data sources, change metadata, etc.

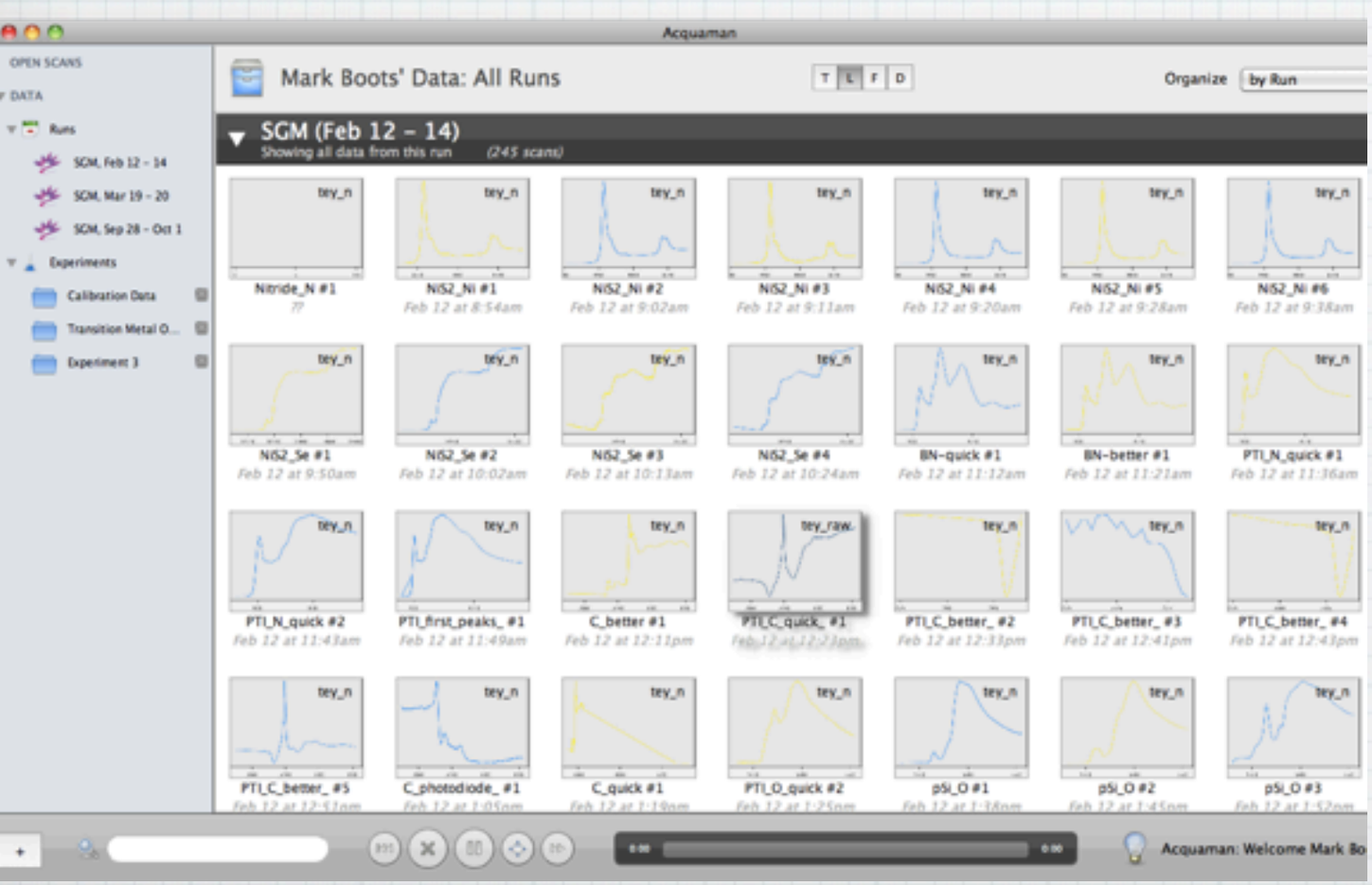



```
AMGenericScanEditor* editor =
    new AMGenericScanEditor();

editor->addScan(scan1);
editor->addScan(scan2);
```

AMScan: UI Views

- **ui/AMDataView**
  - Shows the user their entire database: organized, sortable, browsable
    - Includes all scan objects that share the AMScan table
      - Double-click to edit a scan in a new AMGenericScanEditor
    - Thumbnail view complete; more view options TODO!




```
AMDataView* view =
    new AMDataView
    (AMDatabase::userdb());
```

Applying the dataman module

Custom objects to store in Database?
Subclass AMDbObject,

then `obj->storeToDb()`
`obj->loadFromDb()`

Got some acquiring todo?

Create a new AMScan object

Configure its `rawData()` **[AMDataStore] and then fill it with something groundbreaking**

Create AMRawDataSources to expose your measurements, then `addRawDataSource()`

Set some meta-data

`storeToDb()`

Scan objects to view/plot?
AMScanView, AMGenericScanEditor:
`addScan()`

or: use MPlot directly (*)

Or wait for David's acquisition module...


MASSEY FERGUSON
MASSEY FERGUSON
3) acquisition

Functionality Overview

* 1) Way to interact with the detectors users have available (centers design around user functionality)
  * AMDetector, AMDetectorInfo, AMDetector[Info]Set
* 2) Framework to "get things to happen"
  * AMBeamlineActionItem, AMBeamlineParallelActionsList
* 3) Abstract Interface to describe scanning
  * Configuration, Controller, and Configuration View

Detector Core Classes

AMDetectorInfo

The settings and inherent properties of a detector
(Also a snapshot of those settings)

AMDetector

The means to access the readings and settings for a detector (probably a bunch of AMControls)

AMDetectorSet

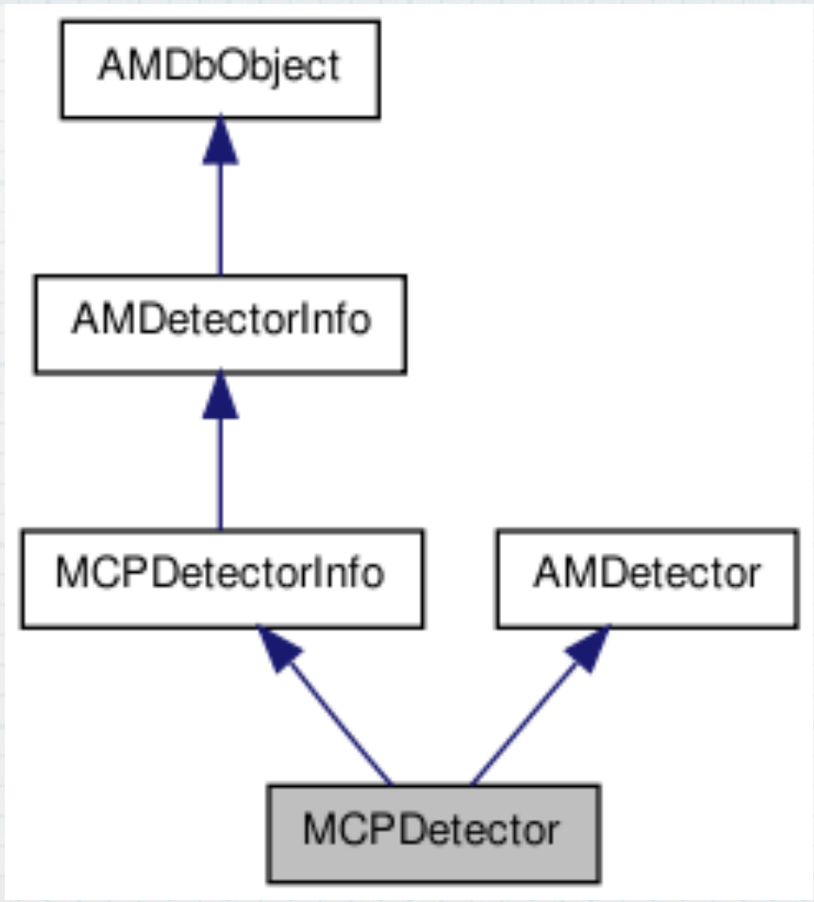
A group of detectors (probably those that can be selected for a particular scan)

AMDetectorInfoSet

A snapshot for an entire group (use it to save or set the state of an AMDetectorSet)

Making a Detector

* 1) Inherit **AMDetectorInfo** and define the **properties and settings** of your detector
* 2) Inherit **AMDetector** and **your detectorInfo** to define how to **read/write** your detector's settings and readings



Let's Make a Detector

Why not a ionization chamber for HXMA (I_T)

* 0) Let's pretend we have an HXMA Beamline Object `HXMABeamline::hxma()`
* 0.1) And it has controls for reading the picoammeter and controlling the voltage

`hxma()->iTransReading()` `hxma()->iTransVoltage()`

* 1) Inherit AMDetectorInfo

```
class IonChamberDetectorInfo : public AMDetectorInfo{
    ...
    double voltageSetpoint();
    void setVoltageSetpoint();
    double voltageSetpoint_;

};
```

Let's Make a Detector

Why not a ionization chamber for HXMA (I_T)

* 2) Inherit AMDetector and your detector info

```
class IonChamberDetector : public AMDetector, public IonChamberDetectorInfo{
    ...
    AMDetectorInfo* toInfo();
    AMControl* readingControl();
    AMControl* voltageControl();
    ...
    void setFromInfo(AMDetectorInfo*);
    ...
};
```

* We'll see how these are used later

What about a Detector View?

* 0) Obviously, it would be really nice to **interact** with your detector in the GUI
* 1) Inherit **AMDetectorView** and make a view

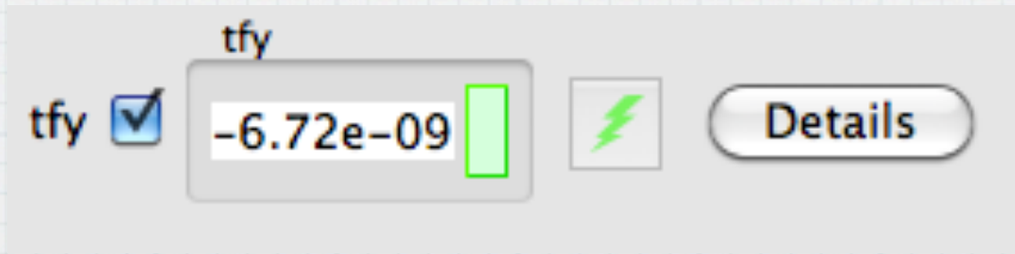
```
class IonChamberBriefDetectorView : public AMBriefDetectorView
```

* 2) Since it's a **"brief view"**, I just want to know what it's reading and whether it's powered on
  * **Reuse** AMControlEditor for the reading
  * **Implement** something for the power state

What about a Detector View?

* 2) (cont'd)

```
readingCE_ = new AMControlEditor(detector_->readingCtrl(), 0, true);
powerState_ = new QToolButton();
powerState_->setIcon(powerOffState_);
powerState_->setEnabled(false);
hl_->addWidget(readingCE_);
hl_->addWidget(powerState_);
connect(detector_, SIGNAL(poweredOnChanged(bool)), this, SLOT(onPoweredOnChanged(bool)));
```



Detector Sets and Views

* 0) Personally, I get tired of coding the same stuff over and over.
  * If the same three detectors are used over and over for a particular scan type, why not group them
* 1) Just make an AMDetectorSet and throw them in there

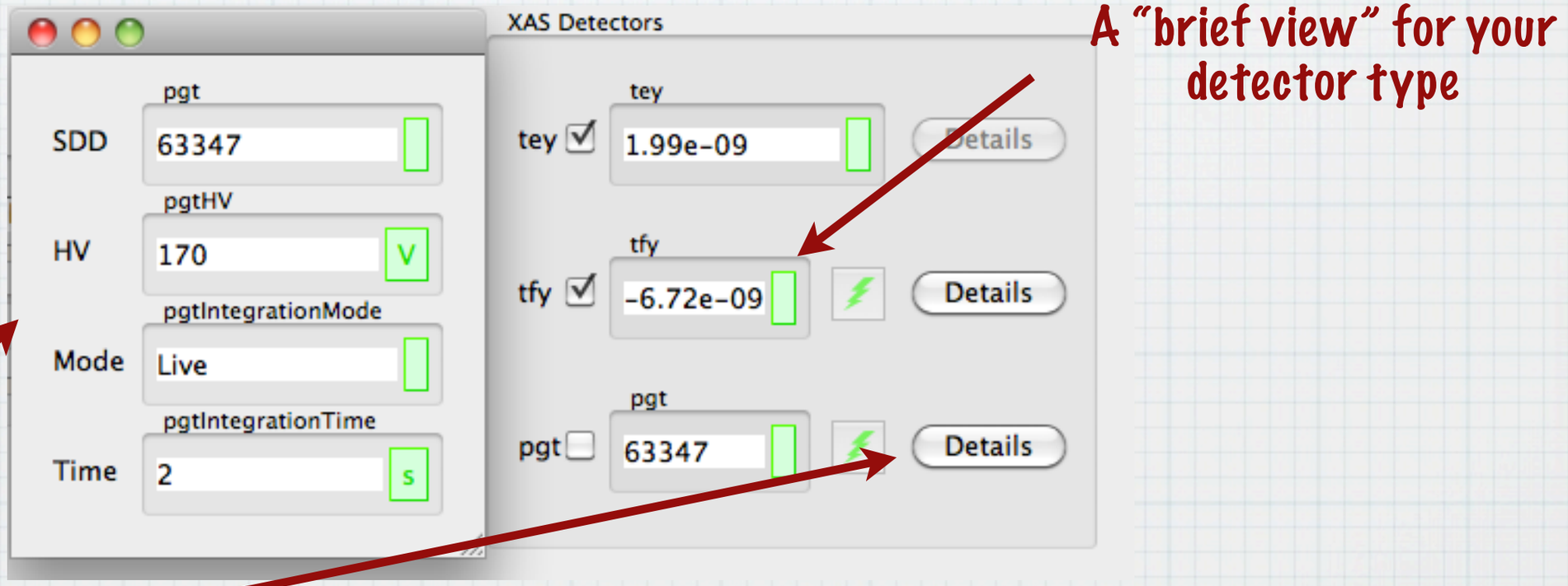
```
AMDetectorSet *xasDetectors_ = new AMDetectorSet();
xasDetectors_->addDetector(hxma()->iTransDetector());
xasDetectors_->addDetector(hxma()->flyDetector());
xasDetectors_->addDetector(hxma()->g32ElementDetector());
```

Detector Sets and Views

* 2) You can register your views for particular detectors

```
AMDetectorViewSupport::registerClass< IonChamberDetector,
   IonChamberBriefDetectorView>
AMDetectorViewSupport::registerClass< IonChamberDetector,
   IonChamberDetailedDetectorView>
```

* 3) Then an AMDetectorSetView can try to display them



A "brief view" for your detector type

And a "detailed view" is accessible if one exists

ActionItem Core Classes

AMBeamlineActionItem

The API defining class which gives an interface to the state machine

AMScanActionItem

AMControlMoveActionItem

AMControlSetMoveActionItem

A list of actions organized into stages. Stages operate in parallel, while the end of one stage starts the next.

AMBeamlineParallelActionsList

AMWorkflowManager(View)

The master list of actions the user wants executed (namely scans or moving to a new sample). All of these classes might actually belong in a Workflow Module.

Starting simple: How to make an action to move a control?

```
// Assume we have a beamline (like sgm() to work with)

AMControlMoveAction *myAction =
    new AMControlMoveAction(sgm()->exitSlitGap());
    // Say you want to move the exit slit gap

myAction->setSetpoint(12.0);
   // Let's tell it to move to 12.0 um
connect(myAction, SIGNAL(succeeded()),
     <someone cool>, SLOT(onActionSucceeded()));
   // Now we'll know when the action has finished successfully

 ... <some time later>

myAction->start(); // Tell the action to start
```

Amping Up: What about that list of actions?

```
// Assume we have a bunch of actions to work with
AMBeamlineParallelActionsList *myList =
    new AMBeamlineParallelActionsList();
    // Setting up a new list
myList->appendStage(); // Set up first stage
myList->appendAction(goToEnergyMidpointAction);
myList->appendStage(); // Set up second stage
myList->appendAction(turnOffUndulatorTrackingAction);
myList->appendAction(turnOffExitSlitTrackingAction);
myList->appendStage(); // Set up third stage
myList->appendAction(goToStartEnergyAction);
myList->appendStage(); // Set up fourth stage
myList->appendAction(changeMonoVelocityAction);
myList->appendAction(changeMonoAccelerationAction);
... <some time later>
myList->start(); // Start these actions
```

Acquisition Core Classes

AMScanConfiguration — The particulars of what should be done for a given scan
AMScanCfg is light, but AMXASScanCfg is bulkier

The "guts" of how to execute a scan — **AMScanController**

Implementation that uses the dacq library under the hood — **AMDacqScanController**

AMScanConfigurationView — GUI to set up a scan configuration (and general API)

AMScanConfigurationView Holder — Trimming around the configuration view to give consistent options (run, queue, etc)

Making a Configuration

So you want to make a configuration

What you'll need

* Beamline
* (Controls)
* (Detectors)

What you'll do

* Inherit the closest option
* Implement the API

What you'll code

* Beamline specific constraints
* Controls options available
* Detectors users can choose

Let's Make a Configuration

Why not an XAS Configuration for SXRMB

What we'll need:

* Let's pretend we have an **SXRMB Beamline Object** `SXRMBBeamline::sxrmb()`
* And it has **controls** for energy, crystal pair, and slits

`sxrmb()->energy()` `sxrmb()->crystal()` `sxrmb()->slits()`

* And the applicable **detectors** (tey, tfy, and sdd)

`sxrmb()->tey()` `sxrmb()->tfy()` `sxrmb()->sdd()`

Let's Make a Configuration

Why not an XAS Configuration for SXRMB

What we'll do:

* There's already an AMXASScanConfiguration, so let's **inherit** it

```
class SXRMBXASScanConfiguration : public AMXASScanConfiguration
```

* **Implement the API** (pure virtual functions)

```
createCopy(){
...}

createController(){
<we'll get to this>}
```

Let's Make a Configuration

Why not an XAS Configuration for SXRMB

What we'll code:

* Beamline **specific** constraints (energy range)

(parent class instantiates an AMXASRegionsList named regions_)

```
SXRMBXASScanConfiguration:: SXRMBXASScanConfiguration{
    regions_->setEnergyControl(sxrmb()->energy())
}
```

* Interface to **controls** and **detectors**

```
double slitWidth()
void setSlitWidth(double)
```

```
AMDetectorInfoSet detectorCfg()
void setDetectorCfg(AMDetectorInfoSet)
```

Making a Controller

So you want to make a controller

What you'll need

* Beamline
* Configuration
* Scan Class

What you'll do

* Inherit the closest controller option
* Implement the API

What you'll code

* Creation of scan object, etc
* Some beamline initialization
* How to scan (**Whoa!!**)

Let's Make a Controller

Why not the XAS Controller for SXRMB

What we'll need:

* We should already have our **SXRMB Beamline Object** `SXRMBBeamline::sxrmb()`
* And, now, we have a **configuration** as well
  `SXRMBXASScanConfiguration *cfg_`
* And there's **already** an **AMXASScan** class
  `AMXASScan *xasScan_`

Let's Make a Controller

Why not the XAS Controller for SXRMB

What we'll do:

* AMDacqScanController is an existing class and SXRMB uses the dacq library, so let's **inherit** it

```
class SXRMBXASScanController : public AMDacqScanController
```

* **Implement the API** (implementation functions)

```
initializeImplementation(){
...}

startImplementation(){
...}
```

Let's Make a Controller

Why not the XAS Controller for SXRMB

What we'll code:

* Create scan object and data sources

```
xasScan_ = new AMXASScan();
loop over ( the detectors I want to use ){
    xasScan_->rawData()->addMeasurement( detector );
    xasScan_->addRawDataSource(new AMRawDataSource(that raw data))
}
```

* Beamline **specific** initialization

```
AMBeamlineParallelActionsList *xasScanInitActions_;
xasScanInitActions_->appendAction(setCrystalAction);
xasScanInitActions_->appendAction(setSlitsActions);
```

Let's Make a Controller

Why not the XAS Controller for SXRMB

What we'll code:

* How to scan ... whoa, Whoa!, WHOA!!
  * Just wait, we **inherited** AMDacqScanController. **What can it do for us?**
* AMDacqScanController will:
  * Give you a QObject to listen to with **signals & slots** (QEpicsAdvAcq class)
  * Make the **output handler** and manage the threaded communication
  * Place the incoming data into the **scan object's data store**
  * ~~Vacuum the laundry, take out the carpet, and wash the trash ...~~

Let's Make a Controller

Why not the XAS Controller for SXRMB

Really Dave, what do I need to code?

* Pass it a **dacq config file** to load up

```
advAcq_->setConfigFile("my path to config file");
```

* Set the "**detectors**" you want to acquire

```
loop over ( the detectors I want to use ){
    advAcq->appendRecord( pvNameLookup(detector) );
}
```

* Reset the **regions**

```
for(int x = 0; x < pCfg_()->regionCount(); x++){
        if(advAcq_->getNumRegions() == x)
                advAcq_->addRegion(x, pCfg_()->regionStart(x), pCfg_()->regionDelta(x), pCfg_()->regionEnd(x), 1);
        else{
                advAcq_->setStart(x, pCfg_()->regionStart(x));
                advAcq_->setDelta(x, pCfg_()->regionDelta(x));
                advAcq_->setEnd(x, pCfg_()->regionEnd(x));
        }
}
```

Making a Config View

So you want to make a configuration view

What you'll need

* Configuration!
* Beamline

What you'll do

* Really depends
* You can use pre-built stuff
* Or, do it yourself

What you'll code

* Inherit the from general config view
* Use whatever you want to display it

Let's Make a Config View

Might as well finish the XAS suite for SXRMB

What we'll need:

* Yeah yeah, we know we need the **configuration**
  `SXRMBXASScanConfiguration *cfg_`
* And the **SXRMB Beamline** Object might be handy `SXRMBBeamline::sxrmb()`

Let's Make a Config View

Might as well finish the XAS suite for SXRMB

What we'll do:

* AMScanConfigView is an existing class with a very small API, so let's inherit it

```
class SXRMBXASScanConfigurationView :
    public AMScanConfigurationView
```

* Implement the ^very small API

```
configuration(){
   return cfg_;
}
```

Let's Make a Config View

Might as well finish the XAS suite for SXRMB

What we'll code:

(I'm lazy, I'm going to use pre-built classes)

* Something to view our regions

```
AMXASRegionsView *xasRegionsView =
    new AMXASRegionsView(cfg_->regions());
```

* And something to view our controls

```
AMControlEditor *slitGapCE =
    new AMControlEditor(sxrmb()->slits(), ..., true);

connect(slitGapCE, SIGNAL(setpointRequested()), this,
    SLOT(newSlitGapConfig()));
```

Set it to "configurationOnly" mode

Connect to the signal that the configuration value changed

Let's Make a Config View

Might as well finish the XAS suite for SXRMB

What we'll code:

(I'm lazy, I'm going to use pre-built classes)

* And something to view our detectors

```
AMDetectorSetView *xasDetectorsView =
    new AMDetectorSetView(cfg_->xasDetectors());
```

* Chuck your stuff in a layout

```
gl_.addWidget(regionsView_,              1, 0, 2, 3, Qt::AlignLeft);
gl_.addWidget(fluxResolutionView_,       3, 0, 2, 3, Qt::AlignLeft);
gl_.addWidget(trackingView_,             1, 3, 2, 2, Qt::AlignLeft);
gl_.addWidget(xasDetectorsView_,         3, 3, 2, 2, Qt::AlignLeft);
gl_.addWidget(warningsLabel_,            2, 0, 1, 5, Qt::AlignCenter);
```

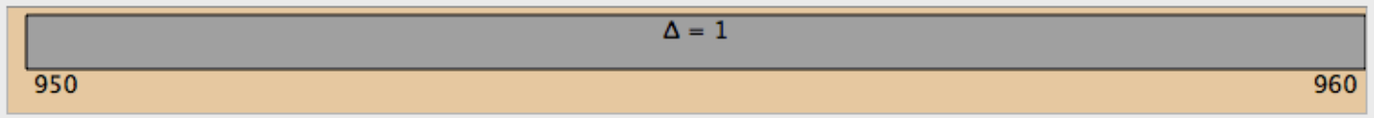
* Throw it into an AMScanConfigViewHolder

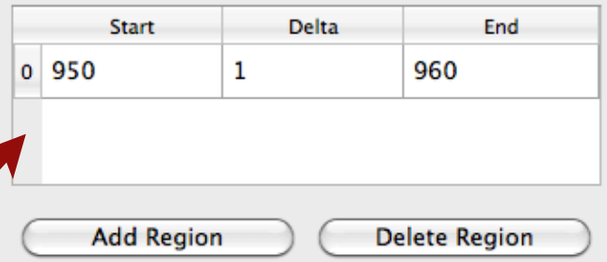
```
xasScanConfigHolder_->setView(mySXRMBXASScanCfgView_);
```

Let's Make a Config View

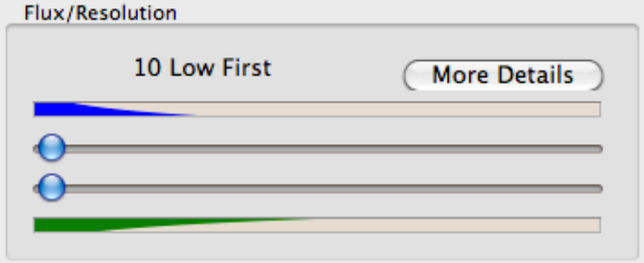
Might as well finish the XAS suite for SXRMB

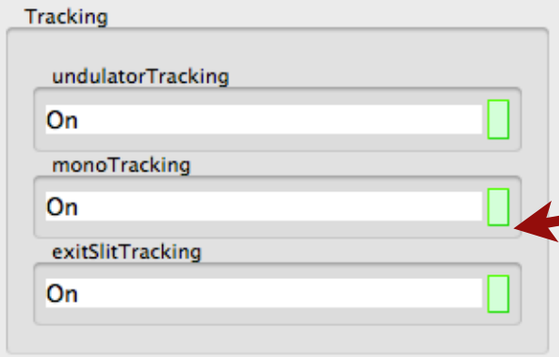
Voilà!





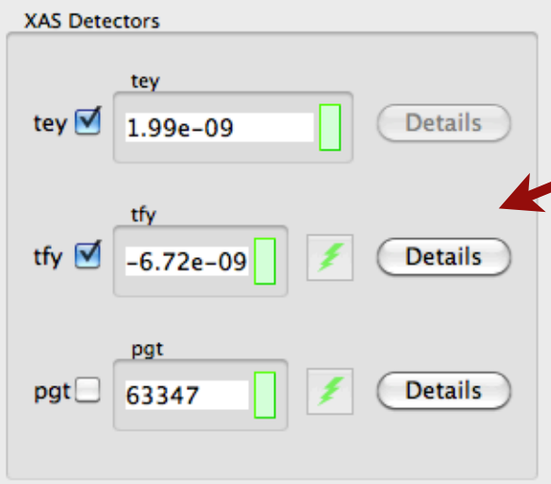
Same regionsView





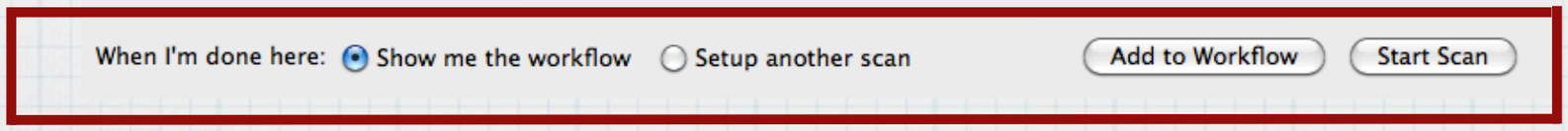
Okay, so this one's mine

Some ControlEditors



Even a detectorSetView

Get this for free with scanConfigViewHolder



Acquisition Classes

What's the big deal?

* 1) Maybe you don't like my regionsView class (tables suck!)
  * Just reimplement the view yourself however you like
* 2) We can make a reasonable model and view for the techniques we share (so users can start quickly on new beamlines)
  * We've got something* for XAS, what about EXAFS?
* 3) Scan controller isn't doing what I want it to
  * Reimplement the acquisition event from the dacq library
  * Make a new controller


SFE 90 Day Bank Bill Futures
CME Eurodollar Futures
SFE 30 Day Interbank Cash Rate Futures
Mar 05
Apr 05
May 05
Jun 05
Jul 05
Aug 05
Sep 05
Oct 05
Nov 05
Jan 06

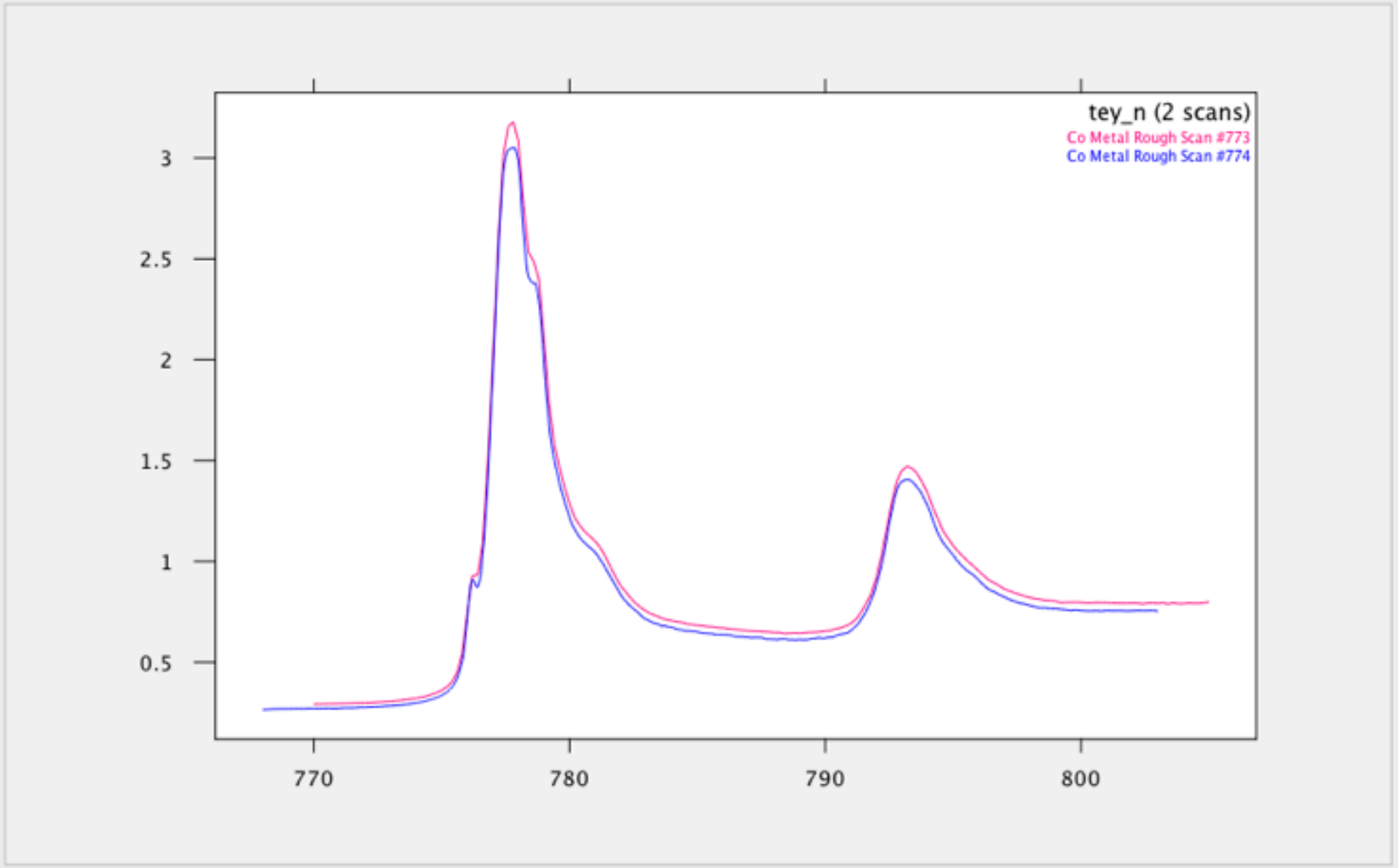
4) Analysis

Everybody needs to do it...

* Helpful to see WHILE COLLECTING DATA...

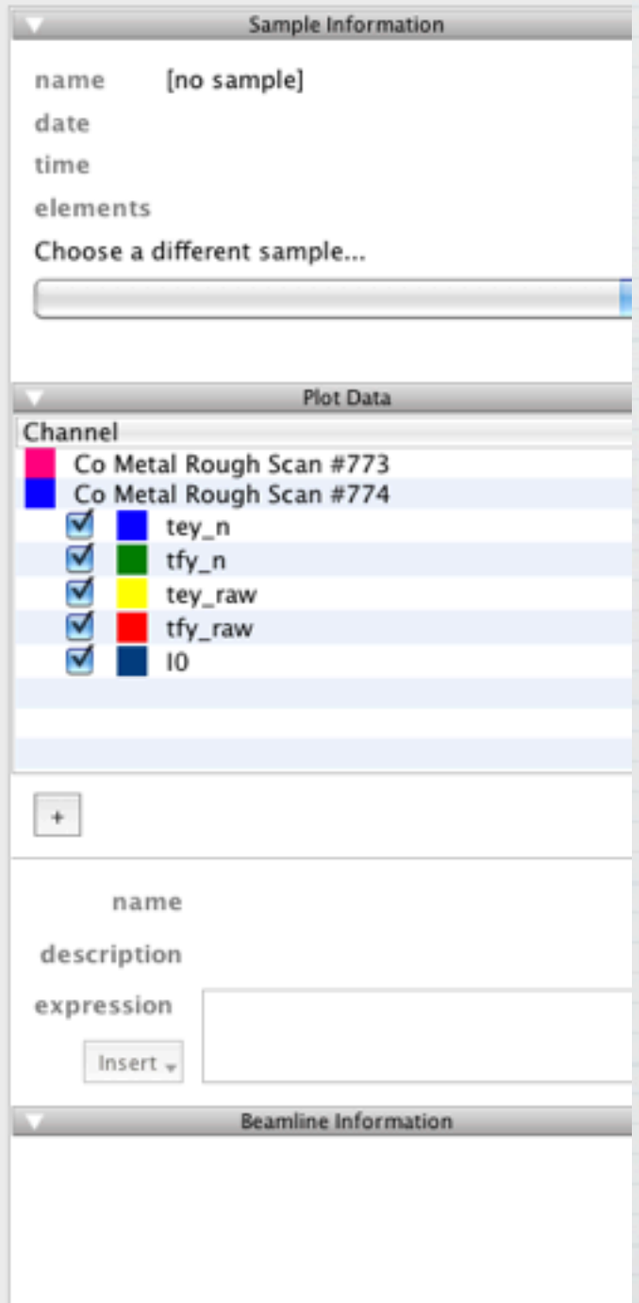
Everybody needs to do it...

* ex: Normalization of a signal to I0 / incident light



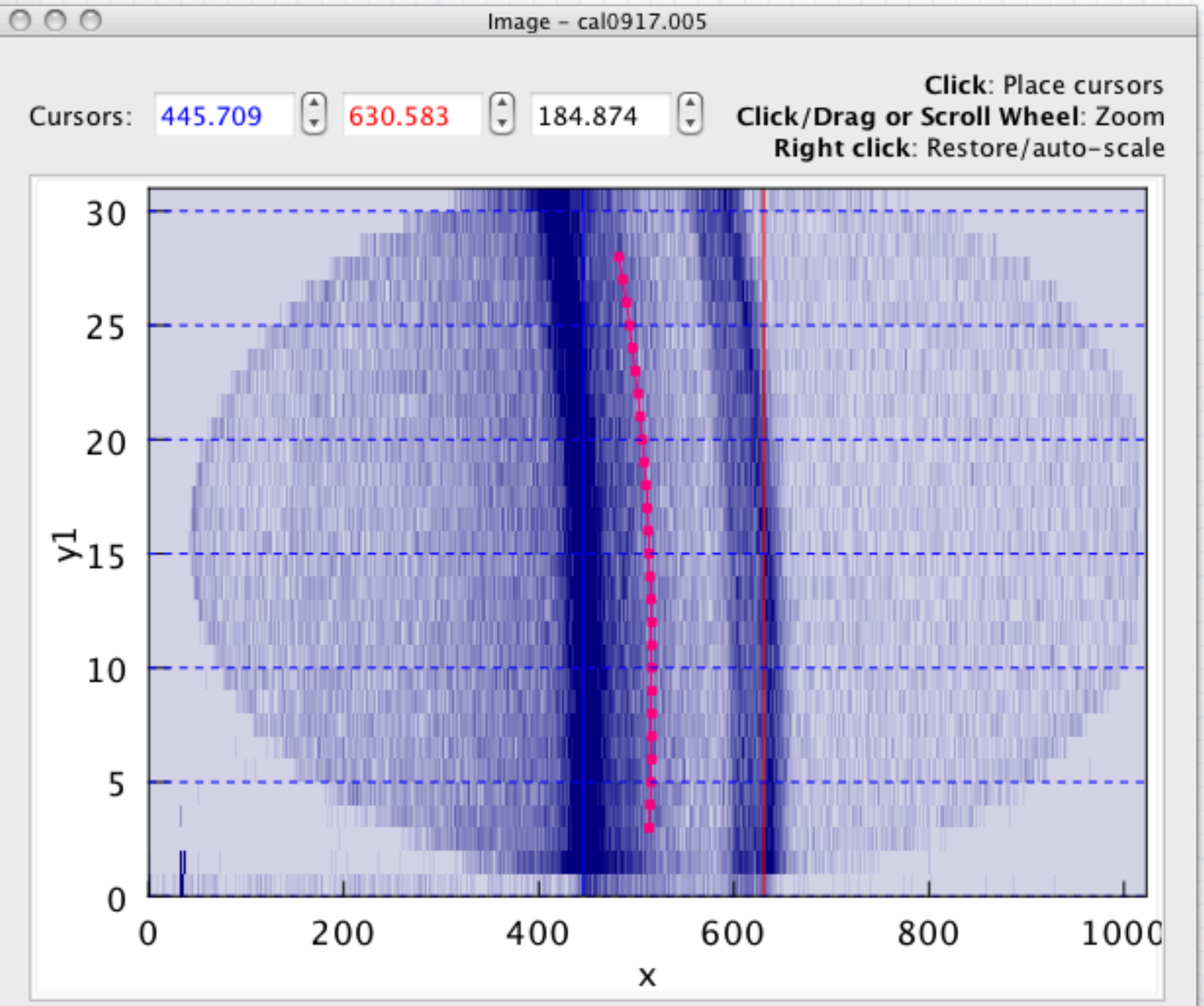
Co Metal Rough Scan #773 tey_n tfy_n tey_raw tfy_raw I0

Co Metal Rough Scan #774 tey_n tfy_n tey_raw tfy_raw I0



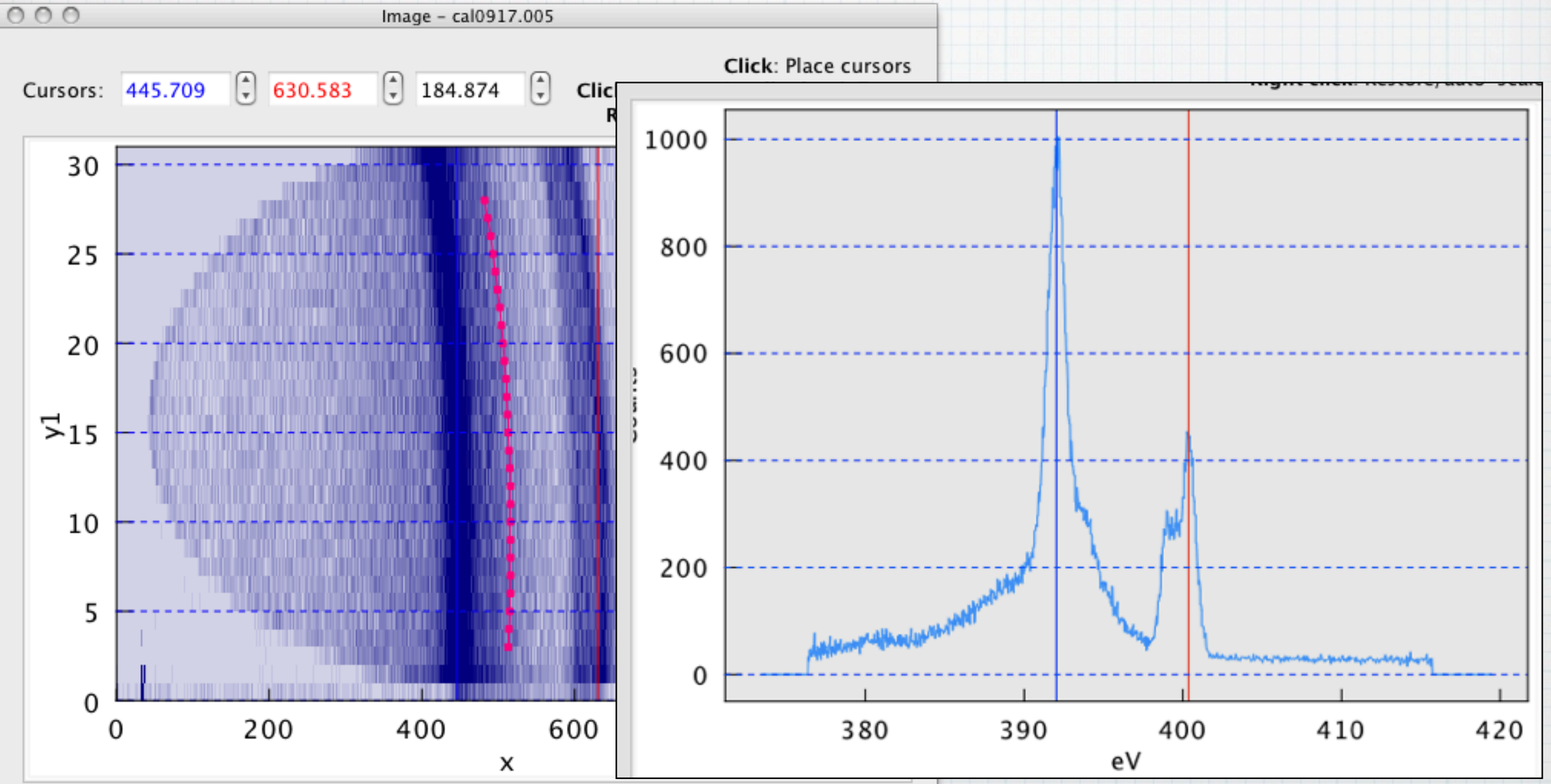
Everybody needs to do it...

* XES Image Processing



Everybody needs to do it...

* XES Image Processing



Everybody needs to do it...

* Helpful to see WHILE COLLECTING DATA (real-time)
  * ex: Normalization of a signal to I0 / incident light
    * More sophisticated methods to remove noise/ account for dirt in SGM beamline
  * ex: Binning a region of a 2D measurement into a 1D dataset
  * XES image processing
  * Region-of-interest binning on a fluorescence micromap scan (per-element / per-edge)

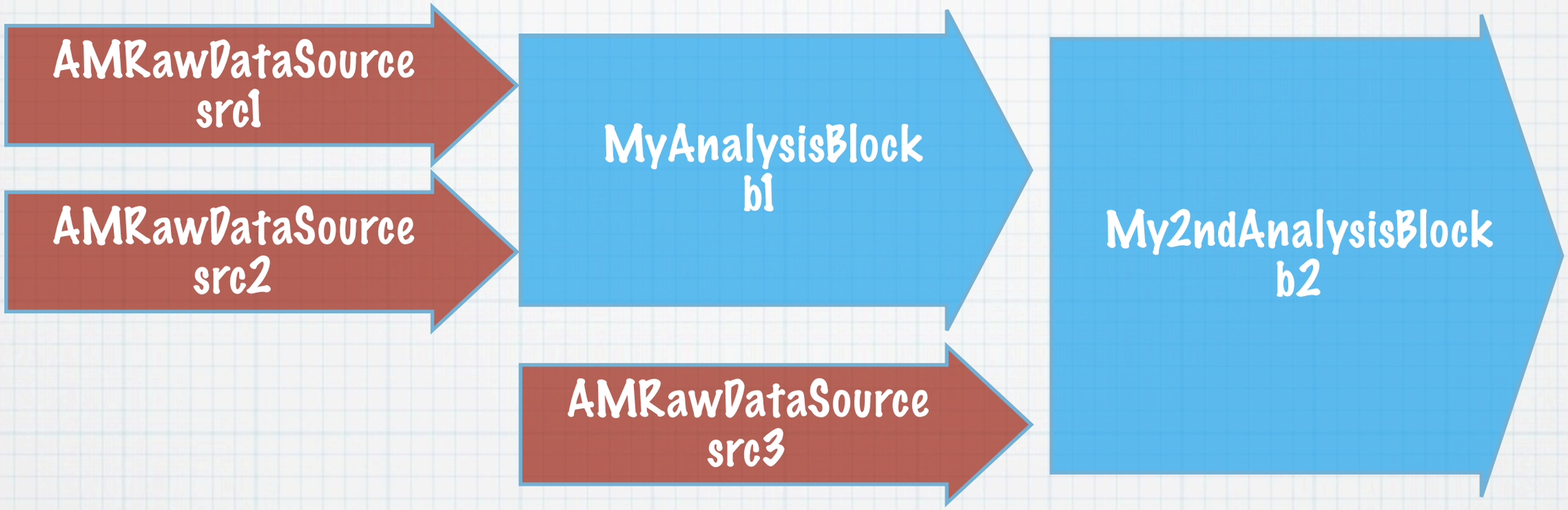
Everybody needs to do it...

* More sophisticated, non-realtime:
  * complex processing of stacks of images to produce strain maps [Vespers]
  * etc.

AMAnalysisBlock

- Analysis framework: uses existing data-representation system in 'dataman' module
- **Chains of "blocks" or filters:**
  - Set a list of `AMDataSource`'s as input
    - Are an `AMDataSource` themselves [for passing to other blocks]
    - Can specify what sets of inputs are allowed
    - Can change size dynamically
      - Constraint: Cannot change # of dimensions dynamically
    - Can change the validity (ex: `state()`, `isValid()`) of their output data dynamically
      - Subsequent blocks wait for valid inputs [handles non-realtime]

AMAnalysisBlock




```
MyAnalysisBlock* b1  = new MyAnalysisBlock(...);
My2ndAnalysisBlock* b2 = new My2ndAnalysisBlock(...);

QList<AMDataSources*> input1;
input1 << src1;
input1 << src2;
b1->setInputDataSources(input1);

QList<AMDataSource*> input2;
input2 << b1;
input2 << src3;
b2->setInputDataSources(input2);
```

AMAnalysisBlock: Implementing your own

- Subclass AMStandardAnalysisBlock
- Store your input data sources in `QList<AMDataSource*> sources_;`
- Store your output axis information (size, units, etc.) in `QList<AMAxisInfo> axes_;`
- Implement

```
bool areInputDataSourcesAcceptable(const QList<AMDataSource*>& dataSources)
```

  - The empty-list must always be acceptable (null data state())
- Implement

```
void setInputDataSourcesImplementation(const QList<AMDataSource*>& dataSources)
```

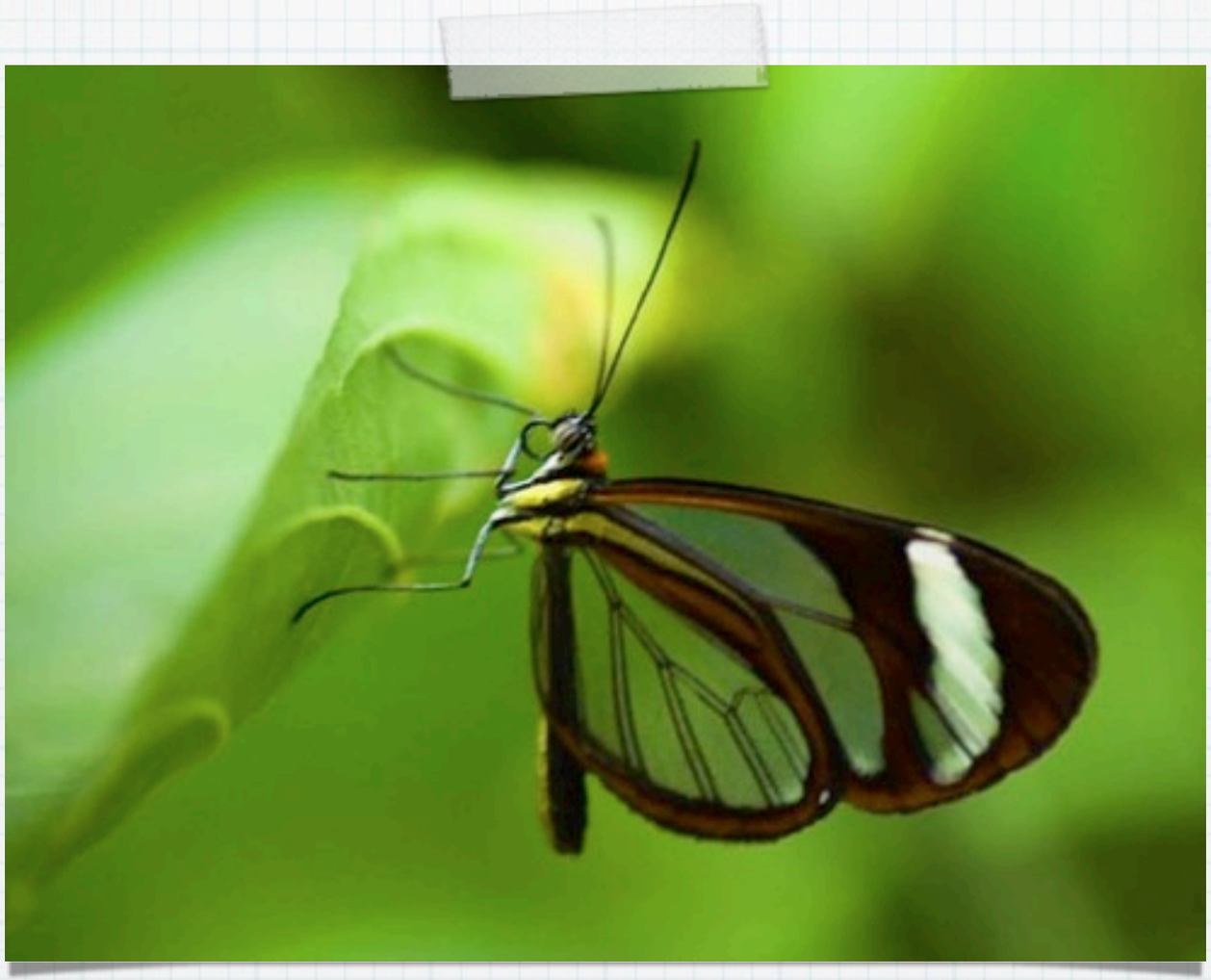
- Call `setState()` whenever the state flags of your data changes (ie: you finish processing, your input data sources are set to <empty>, etc.)
- Implement the AMDataSource API for your output data:
  `value()`, `axisValue()`, etc.
- Optional: Implement

```
QWidget* createEditorWidget()
```

  to create an editor widget that users can use to set any custom parameters, etc.

AMAnalysisBlock:
We've implemented:

AM1DExpressionAB	From a set of 1D input sources: create 1D output based on a math expression
AM2DSummingAB	Takes a single 2D input source: Bins a region horizontally or vertically to get a 1D output
REIXSXESImageAB	Bins a curved XES detector image into a spectrum
Todo: The possibilities...	differentiation, integration, Fourier Transform, Peak Fitting, Fluorescence Map ROI, etc.



5) Visualization

visualization Core Classes

MPlot

Main Plot Object... Contains "items", and has axes, options, etc.
A QGraphicsView framework QGraphicsItem.

A graphics item that draws a XY line or scatter plot

MPlotSeries

MPlotSeriesData

Interface for data that can be used for MPlotSeries.
Set with myPlotSeries->setModel(mySeriesData)

Likewise, but for 2D images and their data

MPlotImage

MPlotImageData

AMDataSourceImageData

Takes an AMDataSource and wraps it to become MPlotImageData. (If the dimensionality is right)

Takes an AMDataSource and wraps it to become MPlotSeriesData

AMDataSourceSeriesData

Applying the visualization module

Use Pre-built Scan Views

`AMScanView`,
`AMGenericScanEditor`

// Uses MPlot internally

Use an MPlot inside an MPlotWidget

Set the axis scales: `setXDataRange()`, `setYDataRangeRight()`, `setYDataRangeLeft()`

or enable auto scaling:
`enableAutoScale(axisFlags)`

Find an MPlotSeriesData object to plot

implement the interface yourself, or use AMDataSourceSeriesData to wrap an AMDataSource

Make an MPlotSeries object

call `setModel()` with the series data object

Add it to the plot

```
plot->addItem(myMPlotSeries)
```

Thanks!