PyBluIce

Modular MX Data Collection Software
for Fully Automated, High Data Rate and Long Duration Experiments

MCE 2021 Workshop @ NSLS-II
March 18, 2021
Mark Hilgart

MX Data Acquisition at GM/CA

Two ID MX Beamlines

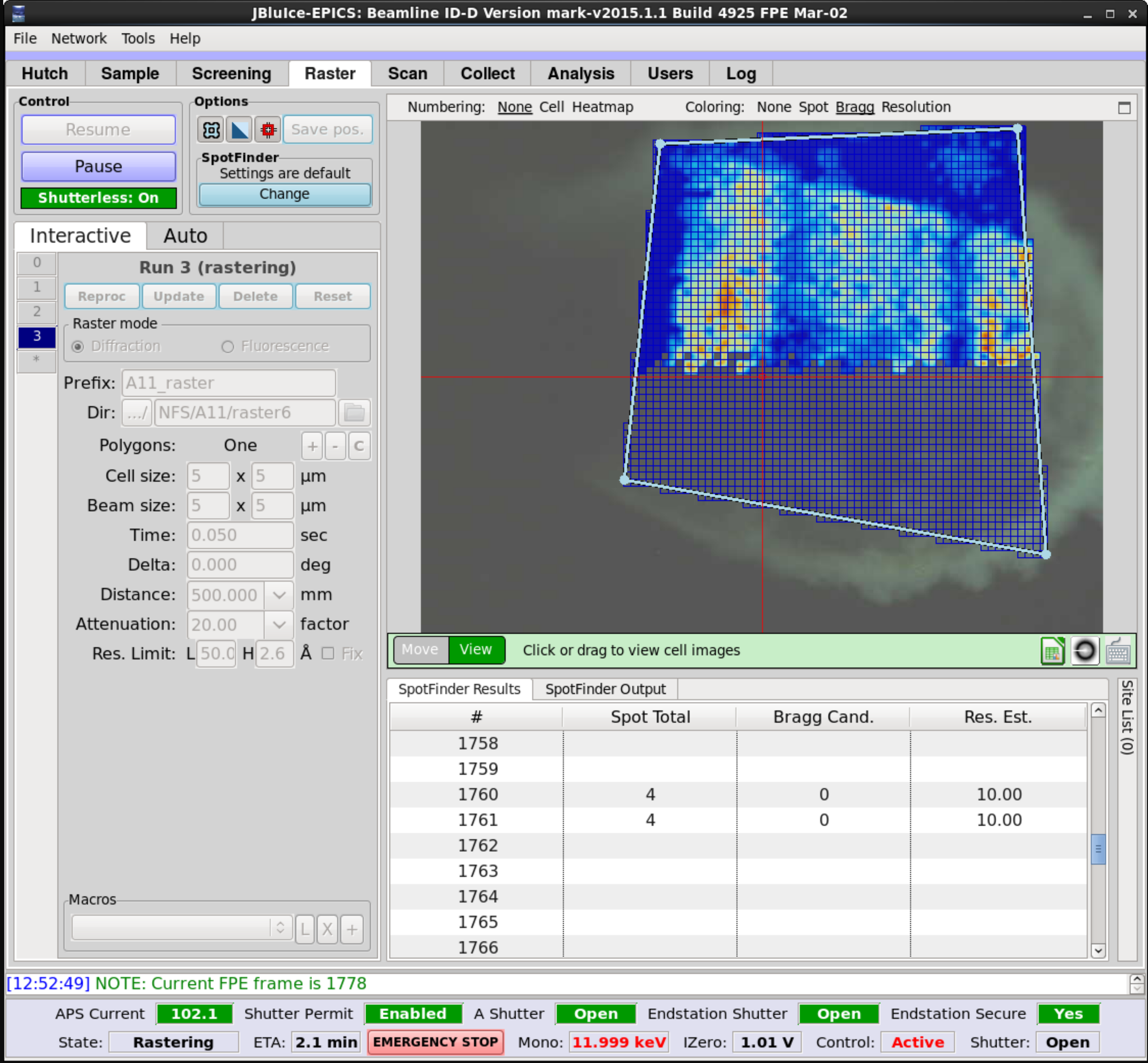
- Sector 23 at the Advanced Photon Source, Argonne National Laboratory

JBluIce is written for and by GM/CA

- Tools are optimized for GM/CA capabilities

Major functions

- Raster, vector and helical collect
- Fluorescence spectrum and edge scan
- Screening
- Analysis



PyBluIce Project

PyBluIce started as a project to bring new features to JBluIce
The benefits we found led us to decide on a full rewrite to Python

The new features we need are

Long Duration Experiments	High Data Rates	Full Automation	More data collection tools
• Million cell raster • Multi-hour injector collect	• 100Hz sustained	• Optical center • Raster-based center • Multi-site strategy • Strategy-based collect	• Several collect modes • Multiple tools per mode

PyBluIce Project

PyBluIce features driving changes

- Long Duration Experiments
- High Data Rates
- Full Automation
- Independent Collect Plug-ins

Improvements needed to deliver these features

Performance

- Responsive GUI while viewing collection plans and results for million frame collections
- Viewing, processing and analysis at hours-long sustained high data rates

Modularity

- Independent collect plugins prevent too much complexity in shared components
- Complex operations need a simple interface for automation

Reliability

- Automation needs to run all day long and recover from errors

PyBluIce Project

PyBluIce features driving changes

- Long Duration Experiments
- High Data Rates
- Full Automation
- Independent Collect Plug-ins

Technical Changes Needed

Current	Change	Benefit
Java & SWT	Python3 & Qt5	Concise and efficient
EPICS & SQL	Redis	Fast, atomic, random access data passing
Single process	Thin GUI with HTTP RPC	Non-GUI load completely offloaded
Hardware abstraction simulation	Realistic beamline simulation	Interactive and automatic short & long-duration tests

Why focus on modularity?

Allow more overall complexity

- Remove specifics from shared components
- Clearly define API between components to expose shared data

Speed development

- Reduced dependencies makes changes easier

Enable a larger development team

- Scripts used by the server are short and easy to modify for new tasks
- The same API used by server components is clearly documented and accessible

Tolerate offline hardware

- Individual testing of modules eliminates unnecessary dependencies

Achieving modularity

Independent scripts

- Complex operations are moved to independent scripts with no server imports, only HTTP/Redis API usage
- Separates high-level, unique tasks from core, shared components
- Provides templates for developers to start with

Unit tests

- Enforce modularity by only starting the modules that are needed

Redis

- Passing data via organized, documented Redis DB locations makes debugging easier
- Easier to diagnose and fix broken database links instead of more complex code-based links

Collect plug-ins

- Moving collect mode-specific code to plug-ins contains their complexity

Reliability for more complex tasks

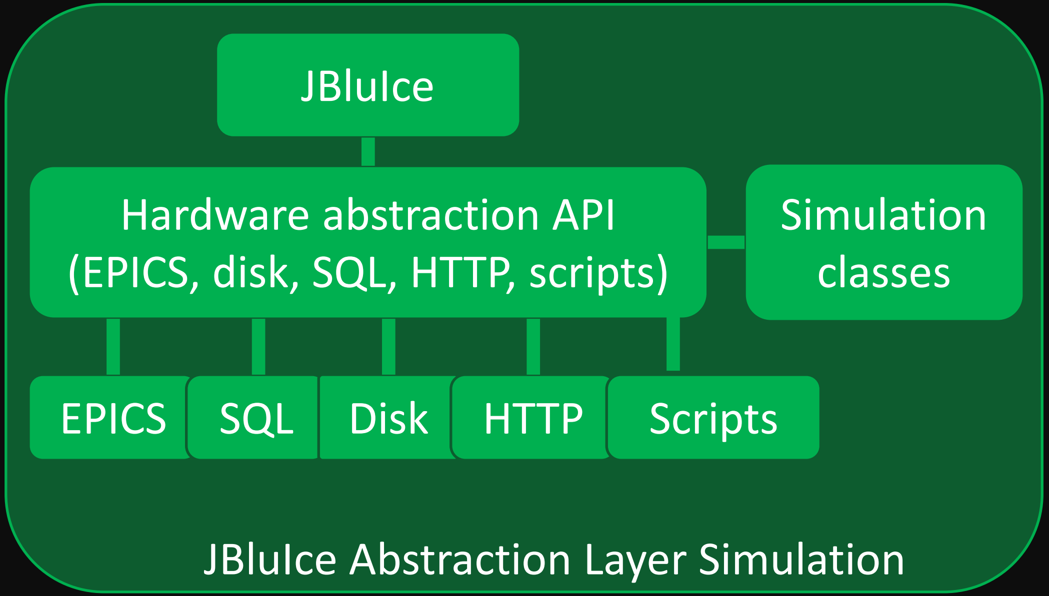
Lights-out automation

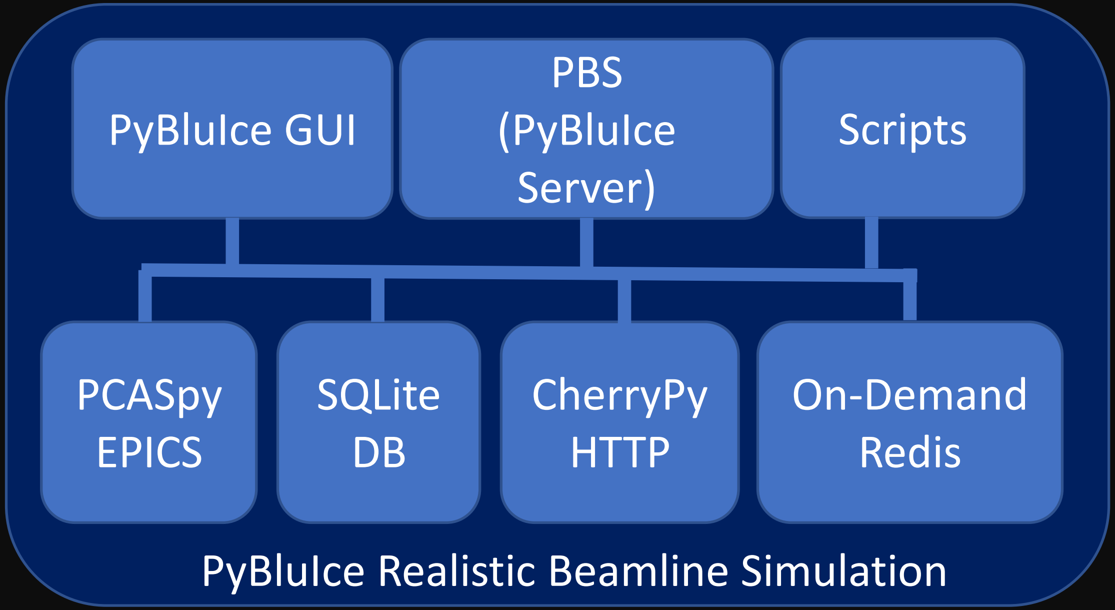
- Detect errors that occur only after repeated or certain patterns of calls
- Recover from hardware errors
- Know when to stop and prompt the user

Reliable scripting interface

- All API options should be tested even if not currently in use

Reliability for more complex tasks: Realistic Simulation





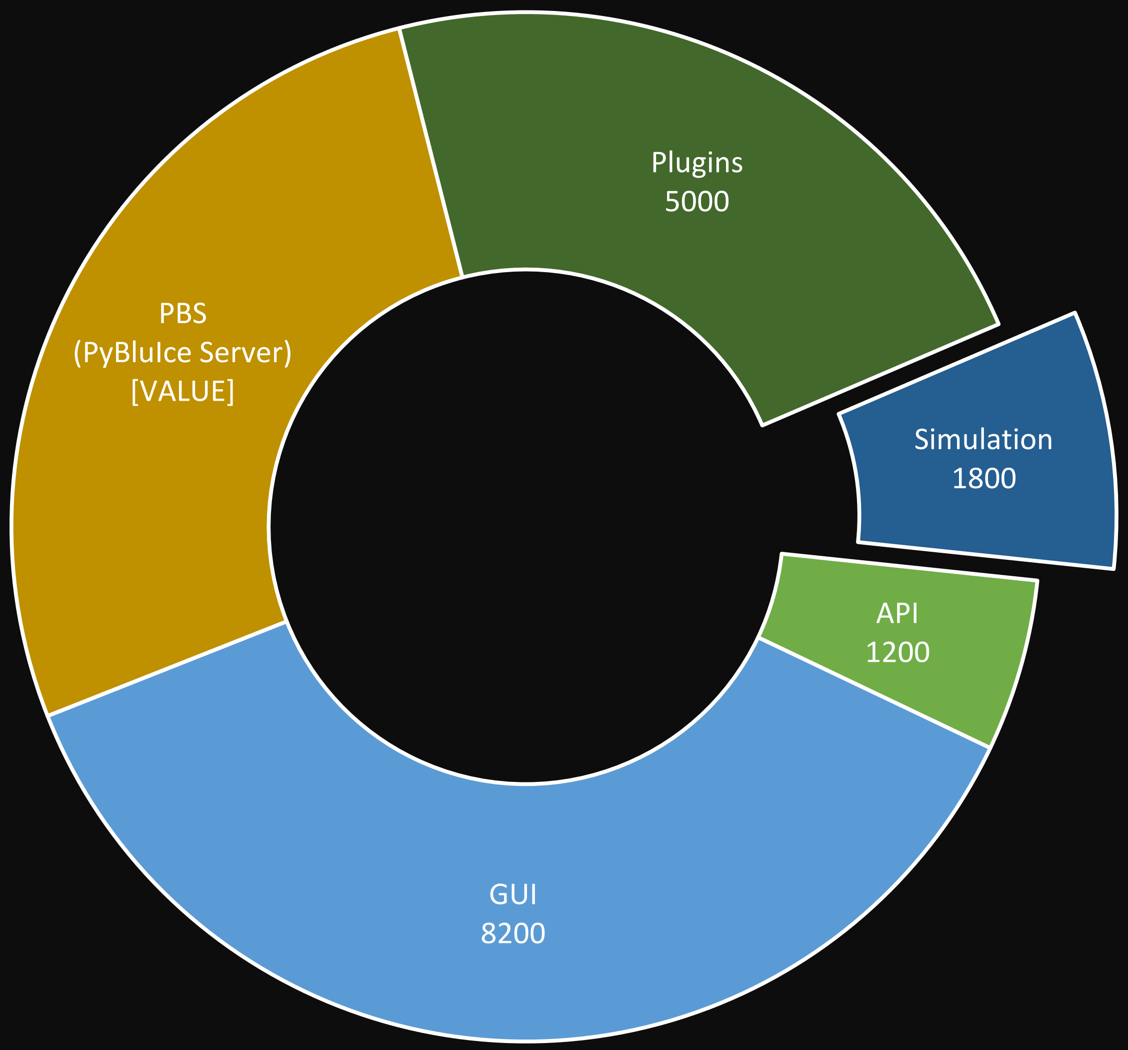
- For JBluIce we wrote an API for each type of hardware device
- This required implementing some part of each protocol e.g. SQL parsing
- High maintenance costs
- Did not achieve offline development

- Python libraries are available to simulate EPICS, SQL and HTTP
- Clients see real EPICS, Redis and HTTP servers
- Enables automated testing and offline development and debugging
- Nearly all operations are currently supported

Lines of Code

Simulation to support all major operations adds 7% to code base

Total size is one fifth of JBluIce's 124,000 LOC



Reaching performance goals

Efficient API for offloading tasks

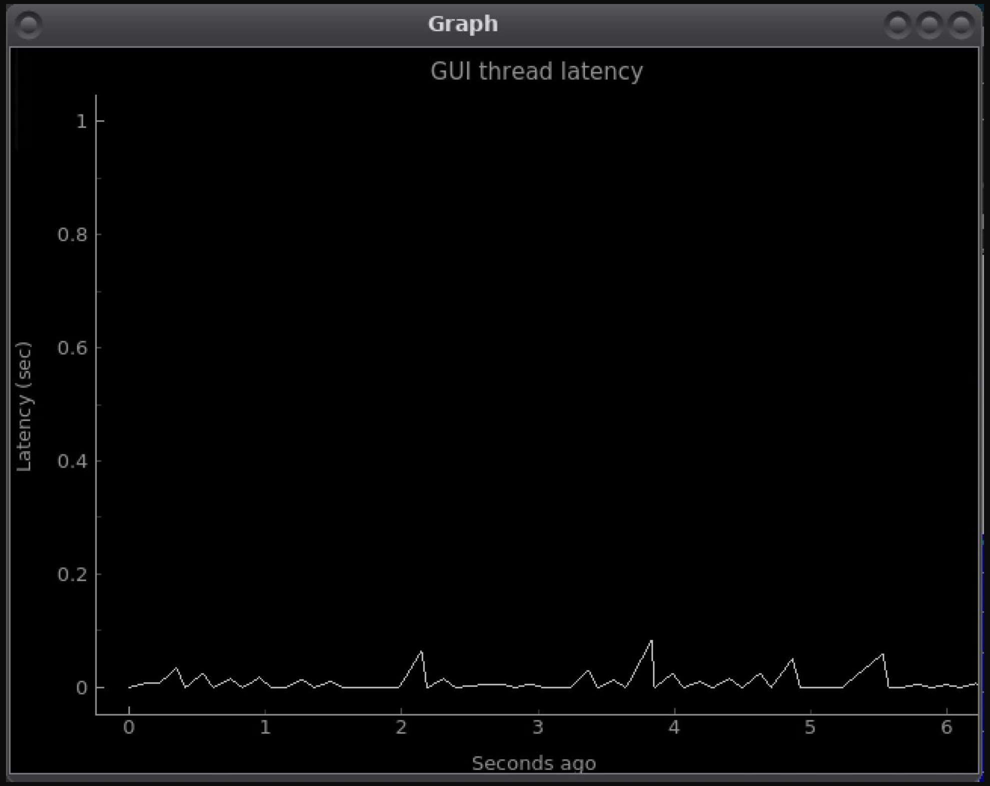
- When offloading is as easy as not, this makes keeping the GUI lean possible
- Slurm tasks also offload processing and drop their results into the database without needing to connect back to the server

High performance PyQt widgets

- pyqtgraph makes video and image viewing at high speed simple
- Lazy tables are more straightforward than in SWT

Modularity and Simulation contribute

- Concise, self-contained methods and modules make refactoring and optimization easier
- Simulation makes regular latency and resource monitoring possible offline



Built-in GUI thread latency viewer

Foundation: Redis API

```
import api.redis_api as ra
ra.run_b.delta_deg.set(0.2, id=1)
```

Setting a data collection parameter using the PyBluIce Redis API import

PyCharm integration

- Tab completion for all levels of field hierarchy
- Control+Q on fields shows their description and formatting

Synchronous, atomic, fast

- Grouping reads and writes into “pipelines” ensures atomicity
- Couple milliseconds per round trip allows for multiple calls within a GUI thread callback
- Can be combined with HTTP calls and the HTTP receiver is guaranteed to see the new values with no delay

Flexible, open

- Above can be done by any script, including external to PyBluIce
- This exposes the inner workings of the servers and makes them accessible for anyone to add on to
- Redis provides linked lists, random-access binary arrays, and other data types

Foundation: HTTP API

```
import api.redis_api as ra
import api.http_api as ha

ra.run_b.delta_deg.set(0.2, id=1)
ha.collect_runs_py(runs=1).post()
```

Starting data collection with a Redis setting followed by an HTTP call

Used for RPC

- Provides reliable parameter passing and immediate or delayed response
- ZeroMQ has protocols that are well-suited to RPC, but HTTP is sufficient, more widely known and has more tools available

Language agnostic

- Any language can either use a library or call curl
- Future web-based interfaces can directly connect to the PyBluIce servers

Auto Tab Preview

Event history

- Shows the user everything automation is doing
- Manual operations and crystal info are shown too
- Previews results: raster grids, camera snapshots, analysis summaries
- Links to full results in other tabs or external browsers
- Sorted by sample ID to include unmounted and manual mount sample events

Global path config

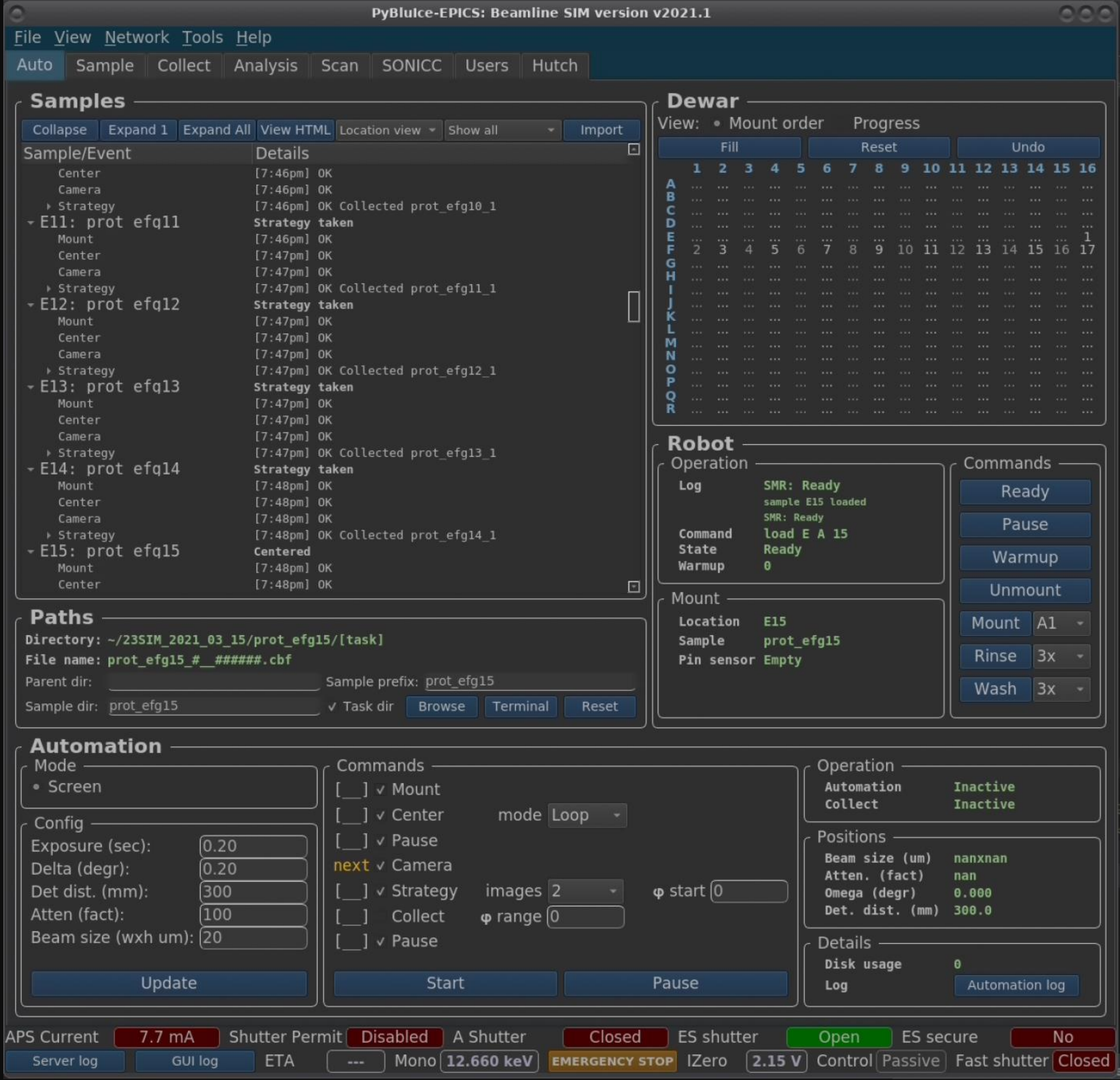
- In automatic mode, operations need a way to determine an output path

Automation queue

- In screening mode, the queue is fixed
- Future modes may allow arbitrary user editing of the queue

Ergonomics

- Compact mode for laptops and large text mode for certain large monitors
- Optional dark mode can reduce eye strain
- Resizable and pop-out widgets make zooming in easy



Collect Tab Preview

Raster plug-in is shown

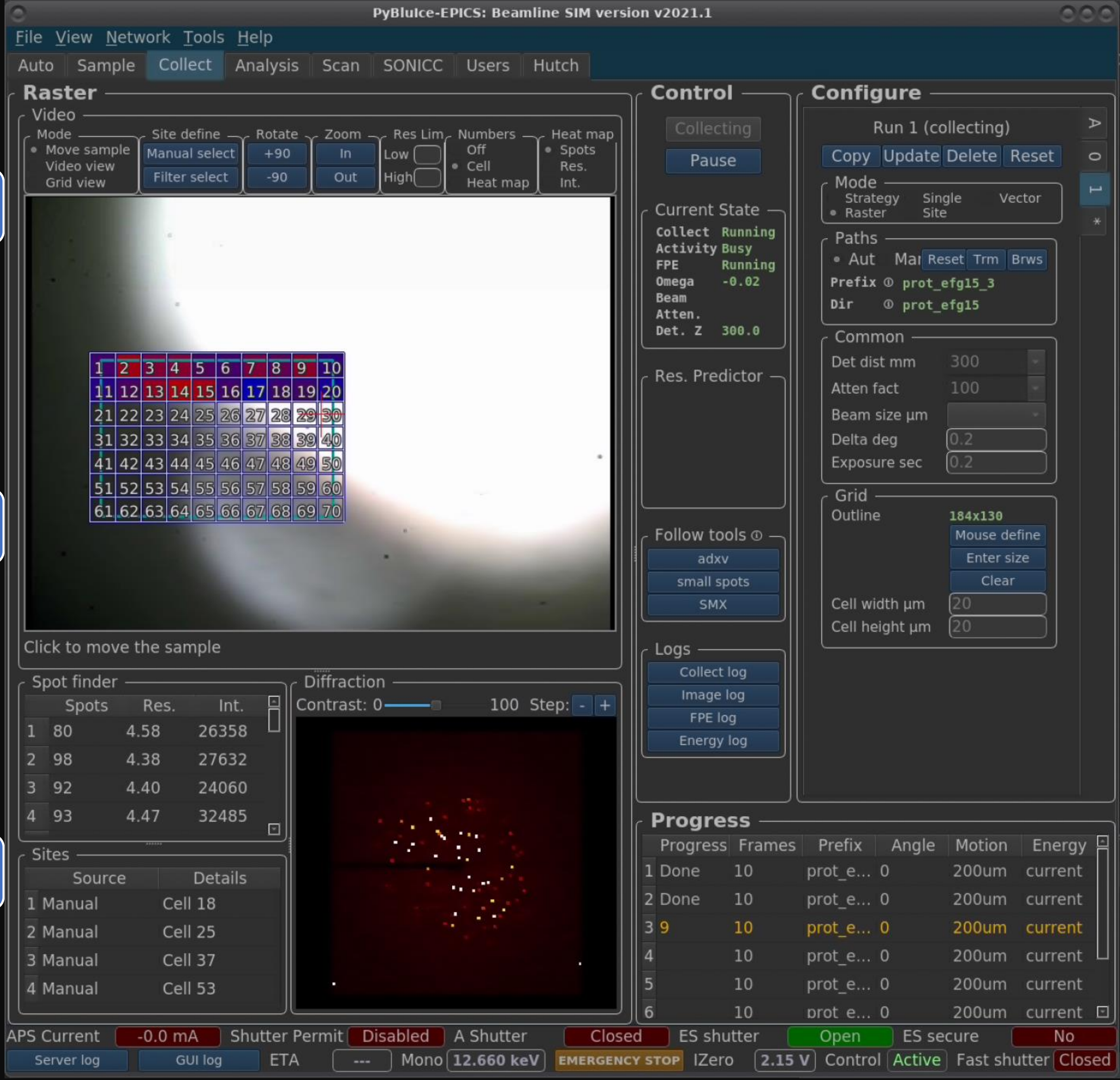
- Collect mode is selected at the top of each run
- Each plug-in:
  - Generates run field widgets
  - Calculates collect commands shown at bottom right
  - Generates widgets on the left half of the tab, though basic widgets like video view have standard versions

10Hz small-spots image viewer

- Server reads image paths from an image “stream”, which is a Redis List that clients watch the length of and read only new items from
- Images are processed at up to 10Hz, and results are saved to another Redis stream in raw binary format with numpy
- Redis+Python+numpy make these steps simple to write and debug
- Processing makes single-pixel spots visible even at the small size shown (algorithm is by David Kissick @ GM/CA)

Shared collect site list

- Site list is a Redis hash of JSON strings
- This is a trade-off making debugging easy, especially with external scripts that may manipulate the list
- Raster mode stores sites in the list, and site mode uses them to collect



Acknowledgements

GM/CA

Janet Smith
Robert Fischetti
Kristin Ahrens
Craig Ogata
Nukri Sanishvili
Michael Becker
Naga Venugopalan
Sergey Stepanov
Oleg Makarov
Qingping Xu
Sudhir Pothineni
Shenglan Xu
David Kissick
Dale Ferguson
Steve Corcoran

Funding

NCI
NIGMS