Simulated EPICS IOCs

Thomas Hopkins

May 16th, 2025

Overview

Background: What are EPICS IOCs?

Motivation: Why do we want simulated EPICS IOCs?

Methods: How do simulated EPICS IOCs work?

Demonstration

Future Plans

Background

EPICS IOCs

EPICS

The **Experimental Physics and Industrial Control System** (**EPICS**) comprises a set of software components and tools that can be used to create distributed control systems. EPICS provides capabilities that are typically expected from a distributed control system:

- Remote control & monitoring of facility equipment
- Automatic sequencing of operations
- Facility mode and configuration control
- Management of common time across the facility
- Alarm detection, reporting and logging
- Closed loop (feedback) control
- Modeling and simulation
- Data conversions and filtering
- Data acquisition including image data
- Data trending, archiving, retrieval and plotting
- Data analysis
- Access security (basic protection against unintended manipulation)

Source: https://docs.epics-controls.org/en/latest/index.html

Brookhaven National Laboratory

Usefulness of an IOC

Input/Output Controllers (IOCs) enable interacting with hardware in a convenient way by providing:

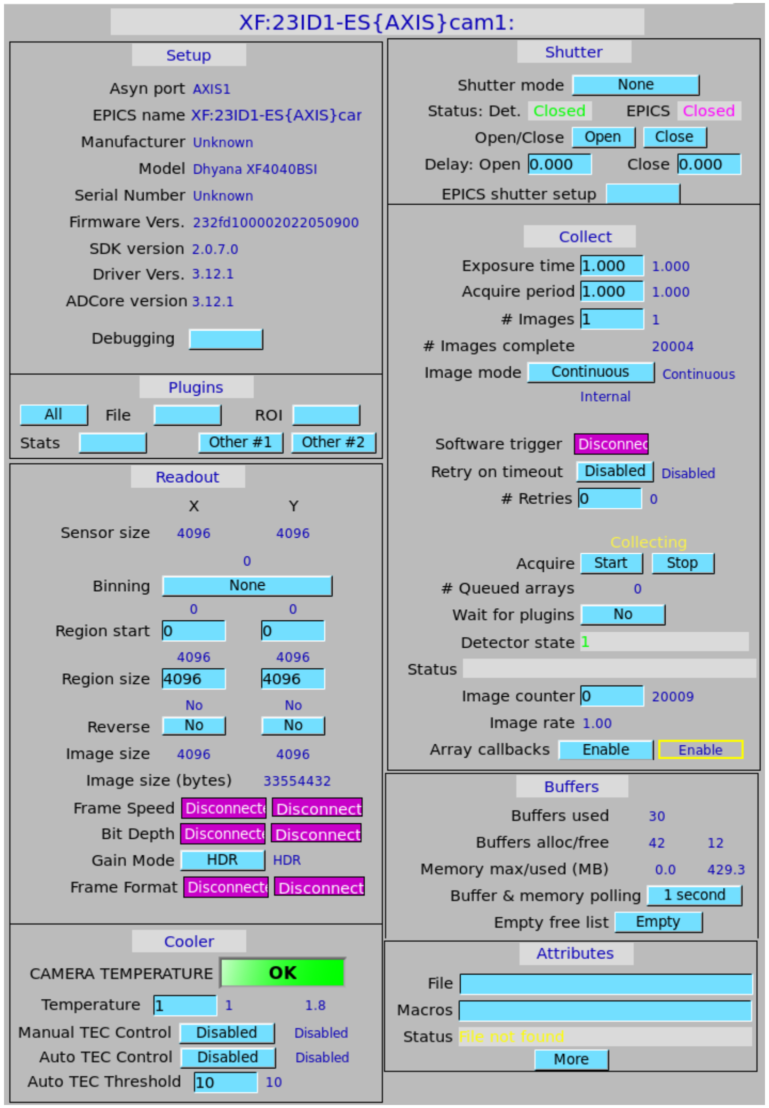
- An abstraction layer
  - A motor controller or a temperature sensor from different vendors may have very different underlying software interfaces, IOCs provide a unified interface for interacting with hardware.
- Network-based control
  - Exposing hardware interfaces over a network allows us to build software downstream of IOCs!
- Modularity
  - Each IOC can be independently developed, configured, and tested.

Example EPICS IOCs – Area Detector

AXIS detector control screen at CSX

Used to control data acquisition. The IOC itself uses the vendor provided SDK to interact with the hardware.

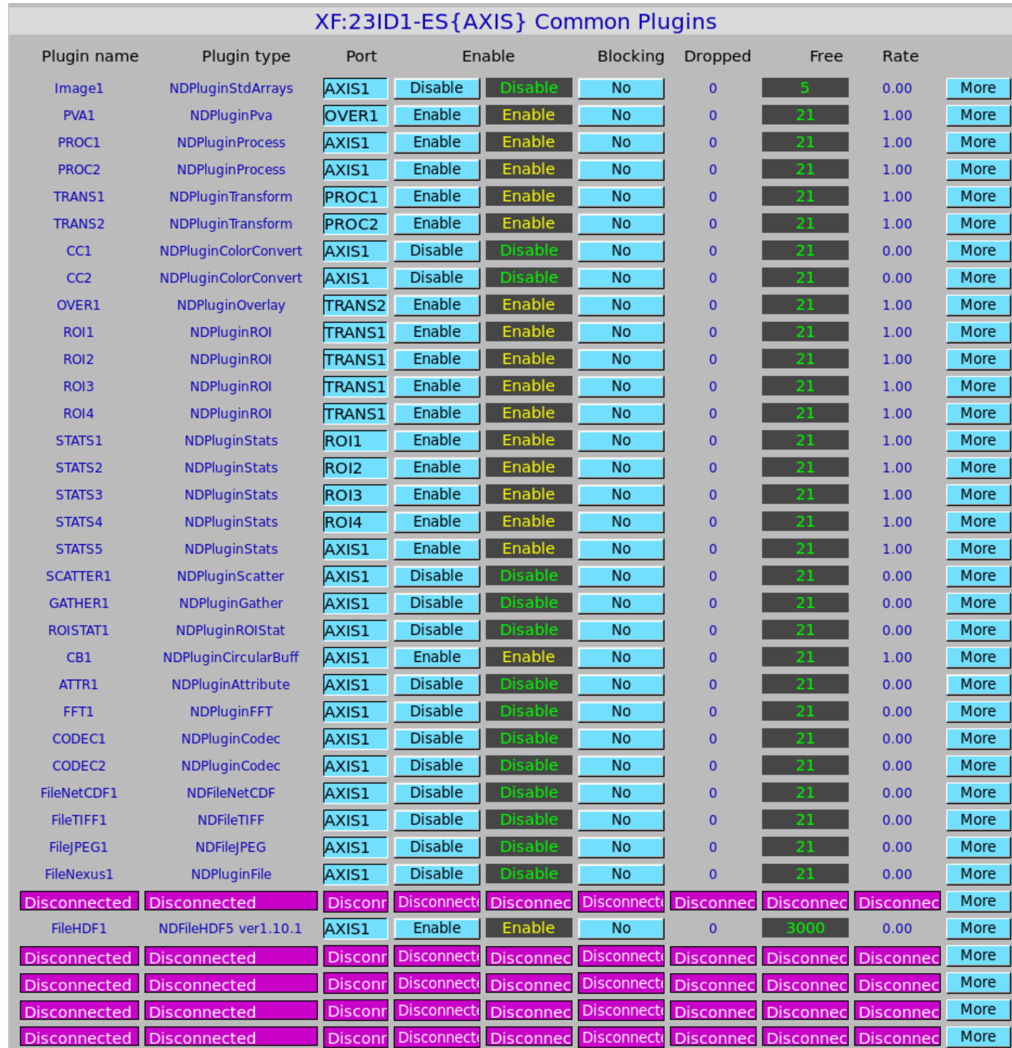
Everything here, except the UI itself, is a Process Variable (PV). PVs describe a single aspect of the process or device under control.



Area Detector Common Plugins

AXIS detector common plugin control screen at CSX.

Used for downstream processing of array data such as regions of interest (ROIs), statistics, transformations, file writing, and much more...



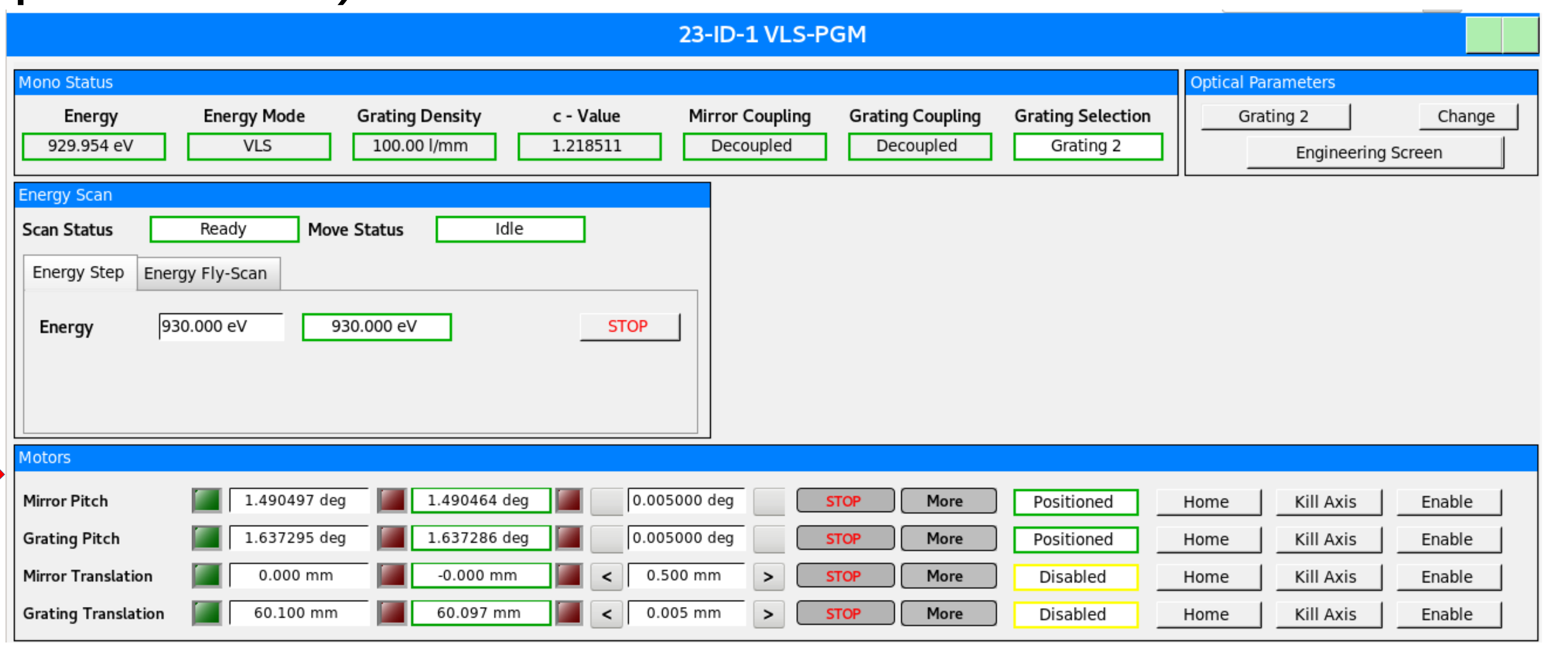
XF:23ID1-ES{AXIS} Common Plugins

Plugin name	Plugin type	Port	Enable		Blocking	Dropped	Free	Rate	
Image1	NDPluginStdArrays	AXIS1	Disable	Disable	No	0	5	0.00	More
PVA1	NDPluginPva	OVER1	Enable	Enable	No	0	21	1.00	More
PROC1	NDPluginProcess	AXIS1	Enable	Enable	No	0	21	1.00	More
PROC2	NDPluginProcess	AXIS1	Enable	Enable	No	0	21	1.00	More
TRANS1	NDPluginTransform	PROC1	Enable	Enable	No	0	21	1.00	More
TRANS2	NDPluginTransform	PROC2	Enable	Enable	No	0	21	1.00	More
CC1	NDPluginColorConvert	AXIS1	Disable	Disable	No	0	21	0.00	More
CC2	NDPluginColorConvert	AXIS1	Disable	Disable	No	0	21	0.00	More
OVER1	NDPluginOverlay	TRANS2	Enable	Enable	No	0	21	1.00	More
ROI1	NDPluginROI	TRANS1	Enable	Enable	No	0	21	1.00	More
ROI2	NDPluginROI	TRANS1	Enable	Enable	No	0	21	1.00	More
ROI3	NDPluginROI	TRANS1	Enable	Enable	No	0	21	1.00	More
ROI4	NDPluginROI	TRANS1	Enable	Enable	No	0	21	1.00	More
STATS1	NDPluginStats	ROI1	Enable	Enable	No	0	21	1.00	More
STATS2	NDPluginStats	ROI2	Enable	Enable	No	0	21	1.00	More
STATS3	NDPluginStats	ROI3	Enable	Enable	No	0	21	1.00	More
STATS4	NDPluginStats	ROI4	Enable	Enable	No	0	21	1.00	More
STATS5	NDPluginStats	AXIS1	Enable	Enable	No	0	21	1.00	More
SCATTER1	NDPluginScatter	AXIS1	Disable	Disable	No	0	21	0.00	More
GATHER1	NDPluginGather	AXIS1	Disable	Disable	No	0	21	0.00	More
ROISTAT1	NDPluginROIStat	AXIS1	Disable	Disable	No	0	21	0.00	More
CB1	NDPluginCircularBuff	AXIS1	Enable	Enable	No	0	21	1.00	More
ATTR1	NDPluginAttribute	AXIS1	Disable	Disable	No	0	21	0.00	More
FFT1	NDPluginFFT	AXIS1	Disable	Disable	No	0	21	0.00	More
CODEC1	NDPluginCodec	AXIS1	Disable	Disable	No	0	21	0.00	More
CODEC2	NDPluginCodec	AXIS1	Disable	Disable	No	0	21	0.00	More
FileNetCDF1	NDFileNetCDF	AXIS1	Disable	Disable	No	0	21	0.00	More
FileTIFF1	NDFileTIFF	AXIS1	Disable	Disable	No	0	21	0.00	More
FileJPEG1	NDFileJPEG	AXIS1	Disable	Disable	No	0	21	0.00	More
FileNexus1	NDPluginFile	AXIS1	Disable	Disable	No	0	21	0.00	More
Disconnected	Disconnected	Disconr	Disconnect	Disconnec	Disconnect	Disconnec	Disconnec	Disconnec	More
FileHDF1	NDFileHDF5 ver1.10.1	AXIS1	Enable	Enable	No	0	3000	0.00	More
Disconnected	Disconnected	Disconr	Disconnect	Disconnec	Disconnect	Disconnec	Disconnec	Disconnec	More
Disconnected	Disconnected	Disconr	Disconnect	Disconnec	Disconnect	Disconnec	Disconnec	Disconnec	More
Disconnected	Disconnected	Disconr	Disconnect	Disconnec	Disconnect	Disconnec	Disconnec	Disconnec	More
Disconnected	Disconnected	Disconr	Disconnect	Disconnec	Disconnect	Disconnec	Disconnec	Disconnec	More

Example EPICS IOCs – VLS-PGM

Variable Line Spacing (VLS) Plane Grating Monochromator (PGM) at CSX

Used to control the optical device via motors (and other parameters).



Ideally...

All of the data sent to and received from hardware at the beamline should go through an IOC.

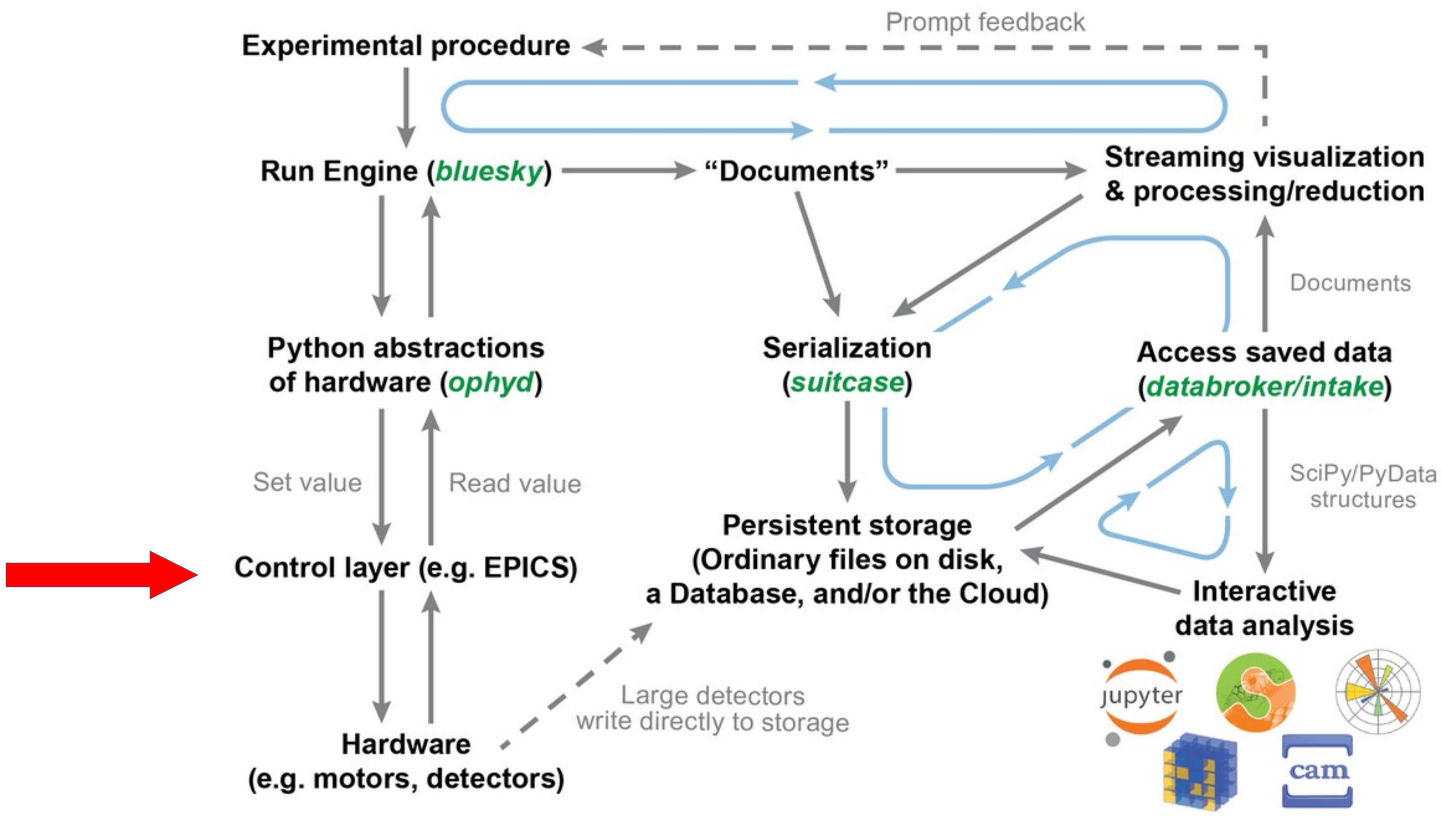
NSLS-II makes heavy use of EPICS Channel Access (CA) as the communication protocol for doing so.

PVAccess (PVA) is a newer protocol but we will not be covering it in this training.

CA is the unifying protocol which we can use to create many downstream services…

Brookhaven National Laboratory

For Instance – Bluesky Ecosystem



Source: https://blueskyproject.io/

Motivation

Simulated EPICS IOCs

Simulation

Beamtime is costly and working with real devices is risky.

- Simulated IOCs allow us to test *downstream services* without having to interact with real hardware.

Continuous integration (CI) and continuous deployment (CD) makes for fewer errors in “production” code.

Software release cycles for these downstream services can therefore be ***more frequent*** and ***reliable***.

Why EPICS CA?

EPICS Channel Access (CA) is the unifying protocol we use to interact with real hardware devices.

This level of the software stack is a great choice for simulation because:

- EPICS IOCs are very close to the real hardware.
- Most services are downstream from these EPICS IOCs.
  - Like AD Plugins, Ophyd, Bluesky, Tiled, Databroker, Prefect, etc.
- We can create general simulators for “kinds” of IOCs rather than try to simulate the diverse array of real hardware.
  - E.g. Every Area Detector IOC (that I am aware of) produces a 2D or 3D array, simulated or not.

Methods

Types of Simulated EPICS IOCs

Disclaimer

Simulated IOCs should ***not*** be deployed to any beamline server unless you have a very specific (and approved) reason for doing so.

The EPICS CA communication protocol does not distinguish between what is “simulated” traffic or “real” traffic which can lead to confusion and costly mistakes.

Simulated IOCs are, however, a very useful development tool, as we shall demonstrate later.

Low vs High Fidelity Simulation

A sufficiently high fidelity simulation could be used for

- Planning experiments
- Constructing new beamlines
- Training data-hungry machine learning models

Low fidelity simulations are more useful for automated testing.

We will be focusing on low fidelity simulation in this hour.

The second hour is showing a high fidelity simulator: XRT.

Brookhaven National Laboratory

Caproto

EPICS Channel Access (CA) in pure Python

- Enables us to build IOCs in Python (rather than C/C++)
- Very useful for simple IOCs

Many example IOCs are available here:

https://github.com/caproto/caproto/tree/main/caproto/ioc_examples

A collaboration between SLAC LCLS and NSLS-II.

Universal Low Fidelity IOC Simulator

One type of low fidelity simulation of an IOC is one that gives a default response for every request.

We call this a “Black Hole IOC” because it is a universal simulator that consumes all PV requests and returns a default response.

- If we do `caget TestPV:cam1:NumImages` it will return `0`
- If we do `caput TestPV:cam1:NumImages 5` it will subsequently return `5` from the same `caget` command above

Important Note: If an IOC such as this were to be deployed at the beamline, it could cause serious issues.

Black Hole IOC

This type of IOC is useful for a few reasons

- It allows us to build up higher fidelity simulators over-time
- It serves as a back-stop for all PV traffic that isn't actively used

The Black Hole IOC is most useful in combination with other *higher* fidelity simulators.

It is built using Caproto and Python!

```
(2025-2.0-py311-tiled) [thopkins1@[redacted] iocs]$ python spoof_beamline.py

*** WARNING ***
This script spawns an EPICS IOC which responds to ALL caget, caput, camonitor
requests.  As this is effectively a PV black hole, it may affect the
performance and functionality of other IOCs on your network.

The script ignores the --interfaces command line argument, always
binding only to 127.0.0.1, superseding the usual default (0.0.0.0) and any
user-provided value.
*** WARNING ***

Press return if you have acknowledged the above, or Ctrl-C to quit.



                    PV blackhole started


[I 14:41:01.865       server:  152] Asyncio server starting up...
[I 14:41:01.865       server:  159] Listening on 127.0.0.1:5064
[I 14:41:01.865       server:  196] Server startup complete.
```

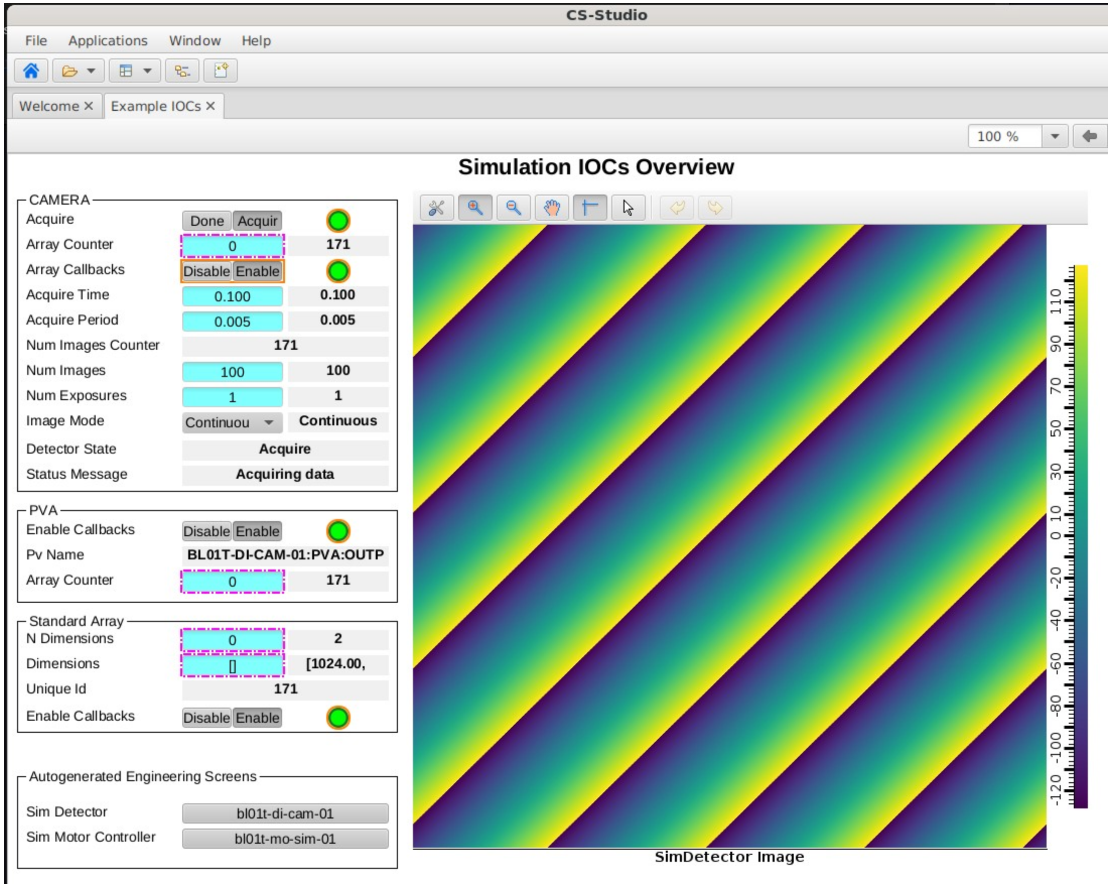
```
(2025-2.0-py311-tiled) [thopkins1@[redacted] iocs]$ caget TestPV:cam1:NumImages
TestPV:cam1:NumImages          0
(2025-2.0-py311-tiled) [thopkins1@[redacted] iocs]$ caput TestPV:cam1:NumImages 5
Old : TestPV:cam1:NumImages          0
New : TestPV:cam1:NumImages          5
(2025-2.0-py311-tiled) [thopkins1@[redacted] iocs]$ caget TestPV:cam1:NumImages
TestPV:cam1:NumImages          5
```

Brookhaven National Laboratory

ADSim IOC

Simulation of a detector built into the Area Detector (AD) framework.

- Provides generic functionality that is similar to most detectors such as capturing a 2D frame at different frequencies.

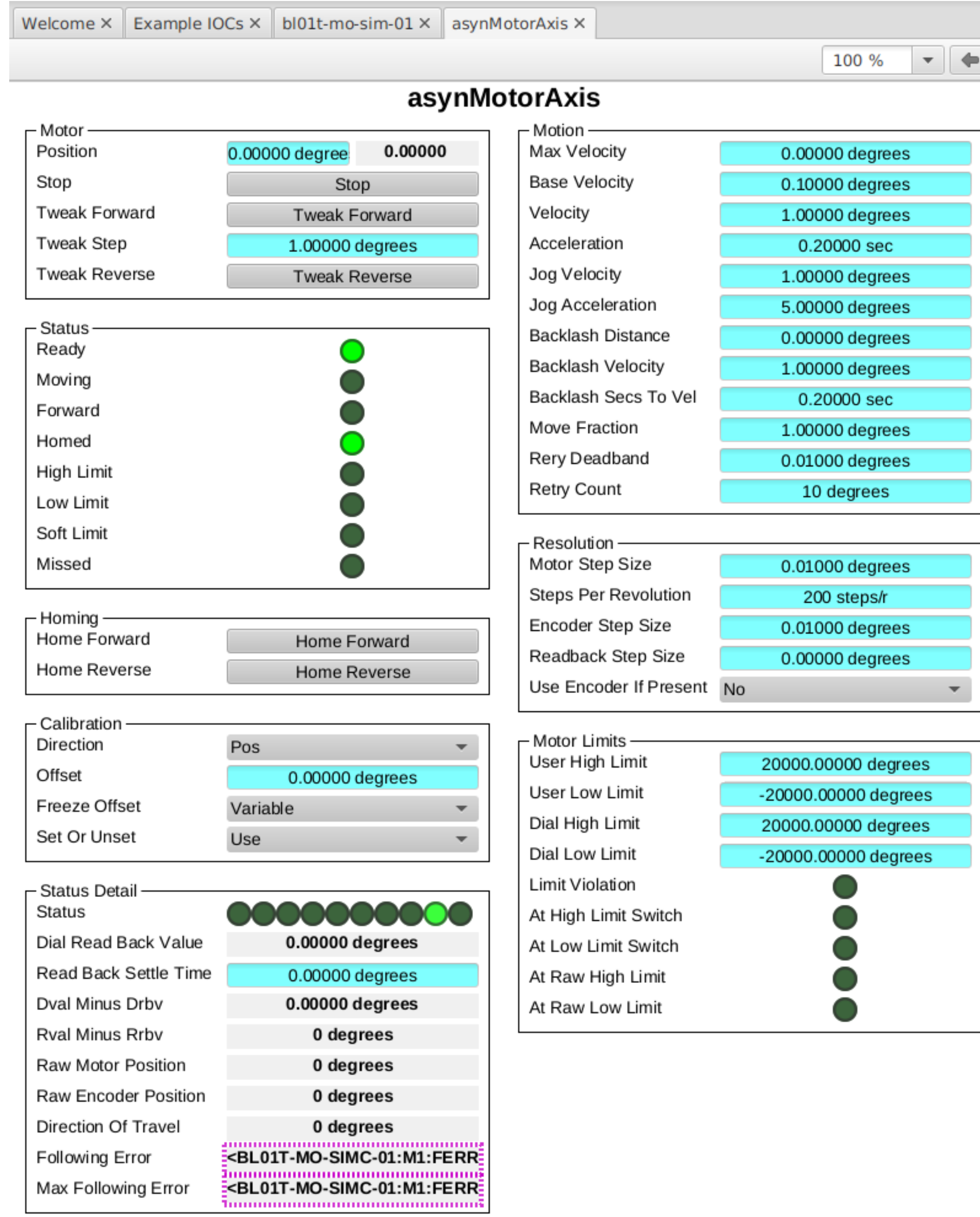


- This is used for unit testing in Ophyd and we would like to expand it to the IPython profiles.
- Enables testing of AD plugins.

Brookhaven National Laboratory

MotorSim IOC

- Simulation of realistic movement for a motor
  - Builds upon EPICS Motor Records, like most motor IOCs do
  - Supports positioning, velocity control, limit handling, and homing.
  - Allows for trajectory scanning along multiple axes.



Brookhaven National Laboratory

Low Fidelity Simulators

The Black Hole IOC, ADSim, and MotorSim are all considered “low fidelity” simulators because they operate in complete isolation from each other.

Changing the motor positions of simulated motors have no effect on the output of ADSim.

Furthermore, the data that is output from ADSim is a fixed pattern that is generated.

Partially Simulated Beamline

The end result of combining:

- Black Hole IOC
- ADSim
- MotorSim

Leads to something that we can use to test basic motor and detector functionality directly from IPython profiles that are used on real hardware.

Furthermore, it allows us to verify that downstream services are compatible with each other when these services are updated.

Containerization of IOCs

Since IOCs can have a large number of dependencies, we use Docker/Podman to deploy IOCs as containers.

- This allows us to compose and deploy multiple IOCs within the same network

We re-use the existing effort by Diamond Light Source: https://github.com/epics-containers/

- Particularly, example-services, which aims to provide a containerized simulated beamline: https://github.com/epics-containers/example-services
- At the EPICS Collaboration Meeting in September 2024, there was an EPICS in containers workshop that goes over the example-services above: https://controlssoftware.sns.ornl.gov/training/2024_EPICS/

CI/CD with Simulated IOCs

We are rolling out CI/CD pipelines via GitHub Actions to "profile_collection" repositories.

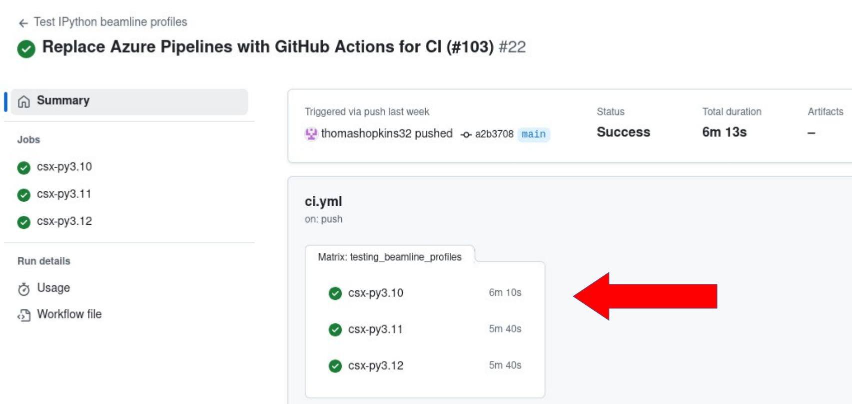
- Start out with Black Hole IOC for basic compatibility and Python interpreter checks

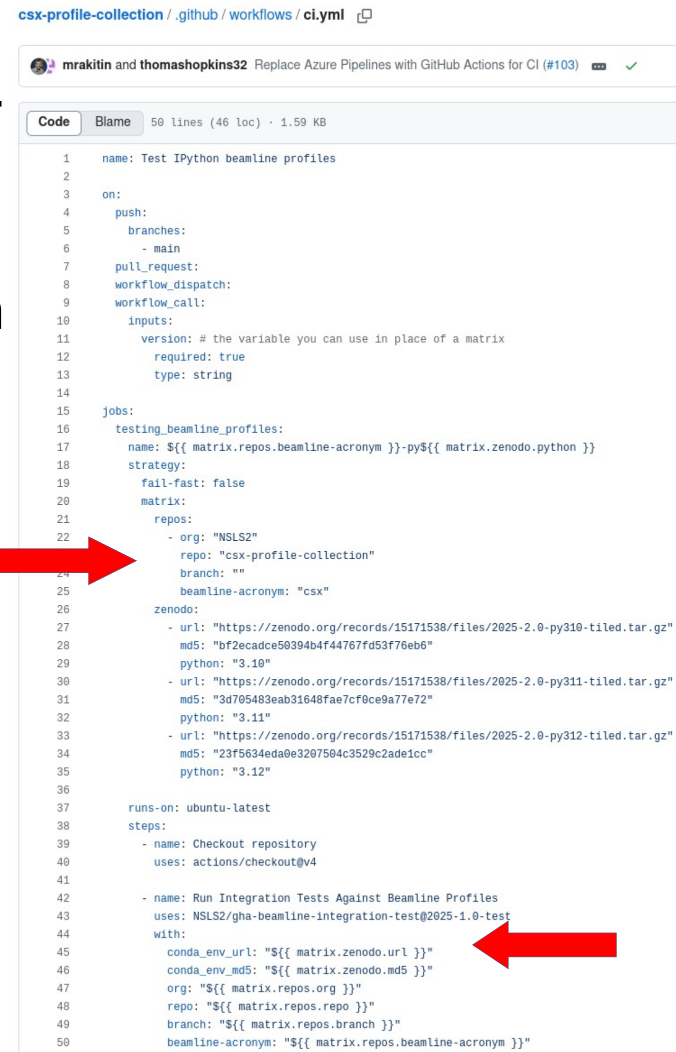
Features	
Black Hole IOC	✅
Downstream Services (Bluesky, Tiled, Redis, etc.)	✅
ADSim	❌
MotorSim	❌
Higher Fidelity Simulations	❌

CSX CI/CD

We use GitHub Actions in csx-profile-collection to test new changes against the Black Hole IOC.

- This allows us to test that the profile can load and the Ophyd objects can connect without error.
- It's a small step in the process toward higher fidelity automated integration testing.





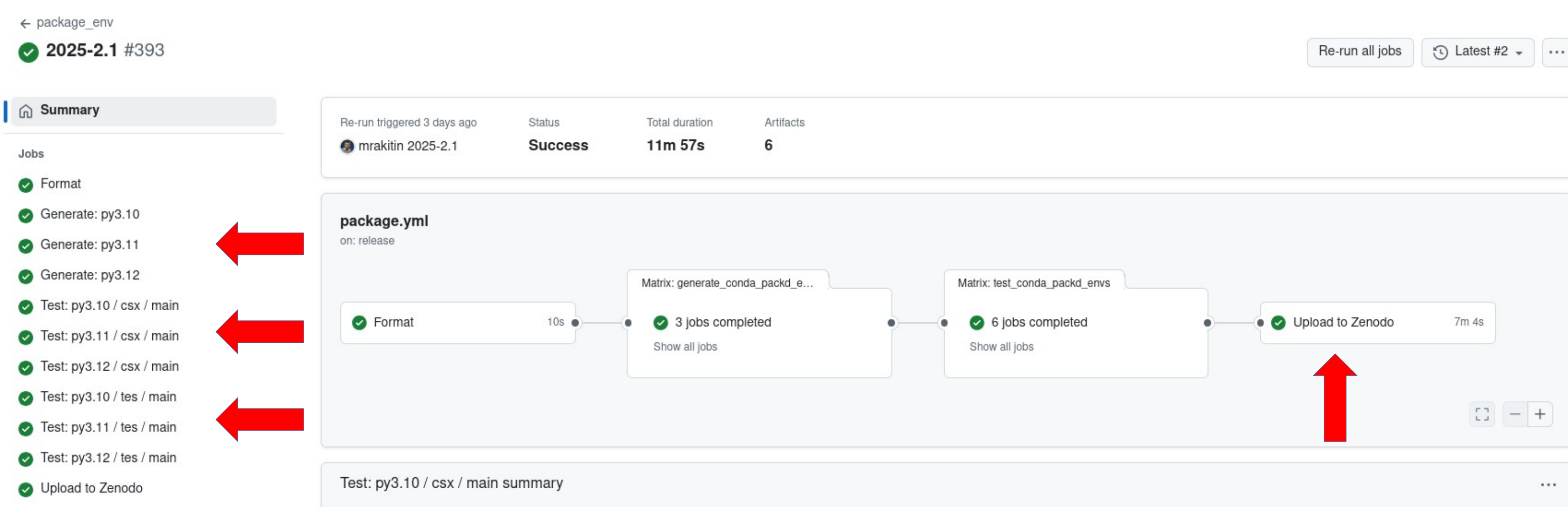

```yaml
name: Test IPython beamline profiles

on:
  push:
    branches:
      - main
  pull_request:
  workflow_dispatch:
  workflow_call:
    inputs:
      version: # the variable you can use in place of a matrix
        required: true
        type: string

jobs:
  testing_beamline_profiles:
    name: ${{ matrix.repos.beamline-acronym }}-py${{ matrix.zenodo.python }}
    strategy:
      fail-fast: false
      matrix:
        repos:
          - org: "NSLS2"
            repo: "csx-profile-collection"
            branch: ""
            beamline-acronym: "csx"
        zenodo:
          - url: "https://zenodo.org/records/15171538/files/2025-2.0-py310-tiled.tar.gz"
            md5: "bf2ecadce50394b4f44767fd53f76eb6"
            python: "3.10"
          - url: "https://zenodo.org/records/15171538/files/2025-2.0-py311-tiled.tar.gz"
            md5: "3d705483eab31648fae7cf0ce9a77e72"
            python: "3.11"
          - url: "https://zenodo.org/records/15171538/files/2025-2.0-py312-tiled.tar.gz"
            md5: "23f5634eda0e3207504c3529c2ade1cc"
            python: "3.12"

    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Run Integration Tests Against Beamline Profiles
        uses: NSLS2/gha-beamline-integration-test@2025-1.0-test
        with:
          conda_env_url: "${{ matrix.zenodo.url }}"
          conda_env_md5: "${{ matrix.zenodo.md5 }}"
          org: "${{ matrix.repos.org }}"
          repo: "${{ matrix.repos.repo }}"
          branch: "${{ matrix.repos.branch }}"
          beamline-acronym: "${{ matrix.repos.beamline-acronym }}"
```

Brookhaven National Laboratory

Conda Environment Release Process

CSX's example of a CI/CD pipeline can be re-used to test all of the beamline profiles at NSLS-II.

- When we generate a new Conda environment, it must pass checks against a few reference beamlines before being uploaded.



Brookhaven National Laboratory

CI/CD in Ophyd

Ophyd uses ADSim to test its core AD framework support.

```yaml
      - name: Start Docker
        run: |
          sudo systemctl start docker
          docker version
          docker compose --version

      - name: Create AD data directories
        run: |
          mkdir -p /tmp/ophyd_AD_test/
          python scripts/create_directories.py /tmp/ophyd_AD_test/data1

      - name: Clone epics-services-for-ophyd
        run: |
          git clone https://github.com/bluesky/epics-services-for-ophyd.git ~/epics-services-for-ophyd

      - name: Start docker containers
        run: |
          source ~/epics-services-for-ophyd/environment.sh
          docker compose -f ~/epics-services-for-ophyd/compose.yaml up -d

      - name: Wait for docker containers to start
        run: |
          sleep 20

      - name: Test with pytest
        run: |
          source ~/epics-services-for-ophyd/environment.sh
          pytest -k "${TEST_CL}"
```

epics-services-for-ophyd / services / adsim_ioc / config / **ioc.yaml**

jennmald move port to be consistent with other entries

Code Blame 152 lines (129 loc) · 2.88 KB

```yaml
# yaml-language-server: $schema=https://github.com/epics-

ioc_name: "{{ _global.get_env('IOC_NAME') }}"
description: Example simulated camera for ophyd adsim

entities:
  - type: devIocStats.iocAdminSoft
    IOC: "{{ ioc_name | upper }}"

  - type: ADSimDetector.simDetector
    PORT: cam
    P: "XF:31IDA-BI{Cam:Tbl}"
    R: ":cam1:"
    DATATYPE: 0
    WIDTH: 1024
    HEIGHT: 1024

  - type: ADCore.NDROI
    PORT: ROI1
    P: XF:31IDA-BI{Cam:Tbl}
    R: ":ROI1:"
    NDARRAY_PORT: cam

  - type: ADCore.NDROI
    PORT: ROI2
    P: XF:31IDA-BI{Cam:Tbl}
    R: ":ROI2:"
    NDARRAY_PORT: cam

  - type: ADCore.NDROI
    PORT: ROI3
    P: XF:31IDA-BI{Cam:Tbl}
    R: ":ROI3:"
    NDARRAY_PORT: cam

  - type: ADCore.NDROI
    PORT: ROI4
```

Brookhaven National Laboratory

Demonstration

Showing simulated EPICS IOCs for CSX and CMS

Future Plans

How we plan to use simulated EPICS IOCs

Usability of the Black Hole IOC

Since the Black Hole IOC consumes all PV traffic, we have to have a way to settle conflicts between this IOC and others we plan on running

- Right now, we have a very crude way of doing this and we are working on improving this
- Suggestions are welcome!

Minimizing Friction

Can we get to the point where spinning up a simulated beamline is as easy as clicking a button?

```
$ SIM=1 bsui
Starting simulated beamline...
Starting local tiled server...
Starting local redis server...
Starting local mongodb...
Loading IPython profile...
```

vs.

```yaml
    - name: Configuring defaults (pyOlog, databroker, and kafka)
      shell: bash -leo pipefail {0}
      run: |
        echo "::group::pyOlog configuration"
        wget https://raw.githubusercontent.com/NSLS-II/profile-collection-ci/master/configs/pyOlog.conf -O $HOME/.pyOlog.conf
        cat $HOME/.pyOlog.conf
        echo "::endgroup::"

        echo "::group::Classic databroker v0/v1 configuration"
        databroker_conf_dir="$HOME/.config/databroker"
        databroker_bl_conf="${BEAMLINE_ACRONYM}.yml"
        mkdir -v -p ${databroker_conf_dir}
        wget https://raw.githubusercontent.com/NSLS-II/profile-collection-ci/master/configs/databroker.yml -O ${databroker_cc
        cp -v ${databroker_conf_dir}/_legacy_config.yml ${databroker_conf_dir}/${databroker_bl_conf}
        cat ${databroker_conf_dir}/_legacy_config.yml
        cat ${databroker_conf_dir}/${databroker_bl_conf}
        echo "::endgroup::"

        echo "::group::Tiled profile configuration"
        tiled_profiles_dir="$HOME/.config/tiled/profiles/"
        mkdir -v -p "${tiled_profiles_dir}"
        cat << EOF > ${tiled_profiles_dir}/profiles.yml
        ${BEAMLINE_ENDSTATION:-${BEAMLINE_ACRONYM:-local}}:
          direct:
            authentication:
              allow_anonymous_access: true
            trees:
            - tree: databroker.mongo_normalized:Tree.from_uri
              path: /
              args:
                uri: mongodb://localhost:27017/metadatastore-local
                asset_registry_uri: mongodb://localhost:27017/asset-registry-local
        EOF
        cat ${tiled_profiles_dir}/profiles.yml
        echo "::endgroup::"

        echo "::group::Kafka configuration"
        cat << EOF > kafka.yml
        ---
          abort_run_on_kafka_exception: false
          bootstrap_servers:
            - localhost:9092
          runengine_producer_config:
            security.protocol: PLAINTEXT
        EOF

        echo "SUDO: Placing kafka config in /etc/bluesky"
        sudo mkdir -v -p /etc/bluesky/
        sudo mv -v kafka.yml /etc/bluesky/kafka.yml
        cat /etc/bluesky/kafka.yml
        echo "::endgroup::"
```

```yaml

    - name: Configuring Redis
      uses: supercharge/redis-github-action@1.8.0
      with:
        redis-version: latest

    - name: Configuring Mongo
      uses: supercharge/mongodb-github-action@1.11.0
      with:
        mongodb-version: '4.4'

    - name: Add info DNS entries for local Redis and Mongo
      shell: bash -leo pipefail {0}
      run: |
        echo "::group::Setting up DNS entries"
        # Per https://stackoverflow.com/a/66982842:
        beamline_redis_address="info.${BEAMLINE_ACRONYM}.nsls2.bnl.gov"
        export BEAMLINE_REDIS_ADDRESS="${beamline_redis_address}"
        echo "BEAMLINE_REDIS_ADDRESS=${BEAMLINE_REDIS_ADDRESS}" >> $GITHUB_ENV
        sudo echo "127.0.0.1 ${BEAMLINE_REDIS_ADDRESS}" | sudo tee -a /etc/hosts
        for i in {1..3}; do
            sudo echo "127.0.0.1 mongo${i}.nsls2.bnl.gov" | sudo tee -a /etc/hosts
        done
        cat /etc/hosts
        echo "::endgroup::"

        echo "::group::Testing connections"
        echo "Pinging ${BEAMLINE_REDIS_ADDRESS}..."
        ping -c 5 ${BEAMLINE_REDIS_ADDRESS}
        echo "Telnetting to ${BEAMLINE_REDIS_ADDRESS}:6379..."
        echo "" | telnet ${BEAMLINE_REDIS_ADDRESS} 6379
        for i in {1..3}; do
            echo "Pinging mongo${i}.nsls2.bnl.gov..."
            ping -c 5 mongo${i}.nsls2.bnl.gov
            echo "Telnetting to mongo${i}.nsls2.bnl.gov:27017..."
            echo "" | telnet mongo${i}.nsls2.bnl.gov 27017
        done
        echo "::endgroup::"

    - name: Insert reasonable values for cycle and data session to redis
      shell: bash -leo pipefail {0}
      run: |
        echo "::group::Installing redis-cli and setting cycle and data session"
        sudo apt install redis-tools
        redis-cli -h ${BEAMLINE_REDIS_ADDRESS} set "${BEAMLINE_REDIS_PREFIX}cycle" "\"2025-2\""
        redis-cli -h ${BEAMLINE_REDIS_ADDRESS} set "${BEAMLINE_REDIS_PREFIX}data_session" "\"pass-000000\""
        redis-cli -h ${BEAMLINE_REDIS_ADDRESS} --scan
        echo "::endgroup::"

    - name: Create nsls2 tiled profile
      shell: bash -leo pipefail {0}
      run: |
        echo "::group::Creating NSLS2 tiled profile"
        conda activate $HOME/env
        tiled profile create --name nsls2 http://127.0.0.1:8000
        echo "::endgroup::"

    - name: Start Tiled Server
      shell: bash -leo pipefail {0}
      run: |
        echo "::group::Starting Tiled server"
        conda activate $HOME/env
        export TILED_API_KEY=secret
        echo "TILED_API_KEY=${TILED_API_KEY}" >> $GITHUB_ENV
        nohup tiled serve catalog --temp --api-key=$TILED_API_KEY &
        sleep 10
        echo "::endgroup::"

    - name: Create TLA tiled profile
      shell: bash -leo pipefail {0}
      run: |
        echo "::group::Creating TLA tiled profile"
        conda activate $HOME/env
        command="
        import os
        from tiled.client import from_profile
        tla = os.getenv('BEAMLINE_ACRONYM', 'xyz').lower()
        TLA = tla.upper()
        client = from_profile('nsls2', api_key=os.getenv(f'TILED_BLUESKY_WRITING_API_KEY_{TLA}', 'secret'))
        client.create_container(tla).create_container('raw', specs=['CatalogOfBlueskyRuns'])
        print(client[tla]['raw'])"
        python3 -c "$command"
        echo "::endgroup::"

    - name: Apply auto settings
      shell: bash -leo pipefail {0}
      run: |
        echo "::group::Applying auto settings"
        if [ -f "${CLONED_REPO}/.ci/auto_settings.sav" ]; then
          while IFS= read -r line; do
            if [ -n "$line" ]; then
                echo "Applying $line"
                caproto-put $line
            fi
          done < "${CLONED_REPO}/.ci/auto_settings.sav"
        else
          echo "No auto settings script found in ${CLONED_REPO}/.ci/auto_settings.sav"
        fi
        echo "::endgroup::"

    - name: Activate IPython profile and run tests
```

Rolling Out CI/CD for Beamlines

We plan to use the Black Hole IOC as a starting point for CI/CD testing for all beamline IPython profiles.

- This will enable us to perform very simple runtime checks

Next, we plan on integrating ADSim and MotorSim to automatically test actual Bluesky plans.

More Simulators for Testing

Can we develop higher fidelity simulators for CI/CD testing?

- Instead of having ADSim that tries to simulate area detectors generally, could we implement an ADPilatusSim that is closer to how the Pilatus series of detectors works?
- Some detectors have additional functionality beyond what ADSim provides that could be useful to simulate

Brookhaven National Laboratory

How you can help

- Use Git and push your changes to GitHub
  - If you work on any beamline software, keep your repository up-to-date following Git/GitHub best practices
  - Your code should soon be automatically tested
- Share simulators with us if you have them!
  - We can help integrate them into automated testing pipelines
- Use these tools for development and give us feedback on what works well and what doesn't
  - Where is the most friction? Where is this not useful yet?

References

- EPICS Homepage: https://epics-controls.org/
- Caproto Homepage: https://caproto.github.io/caproto/v1.2.0/
- Black Hole IOC: https://github.com/caproto/caproto/blob/main/caproto/ioc_examples/pathological/spoof_beamline.py
- ADSimDetector: https://github.com/areaDetector/ADSimDetector
- MotorSim: https://github.com/epics-motor/motorMotorSim
- Beamline Integration Test GitHub Action: https://github.com/NSLS2/gha-beamline-integration-test
- EPICS Containers: https://github.com/epics-containers/
- EPICS Collaboration Meeting on EPICS Containers: https://controlssoftware.sns.ornl.gov/training/2024_EPICS/

Thank you to all of the contributors who make projects like this possible! This is a combination of a lot of software that took many years to put together.

Questions?

Brookhaven National Laboratory