APS_BlueSky_tools Documentation

Release 0.0.40+28.g0822f71.dirty

Pete R. Jemian

Jan 03, 2019

Contents:

Various Python tools for use with BlueSky at the APS

- http://nsls-ii.github.io/
- https://github.com/NSLS-II/bluesky

CHAPTER 1

Package Information

author Pete R. Jemian

email jemian@anl.gov

documentation https://APS_BlueSky_tools.readthedocs.io

source https://github.com/BCDA-APS/APS_BlueSky_tools

1.1 Applications

There are two command-line applications provided by APS_BlueSky_tools:

application purpose	
aps_bluesky_tools_plan_catalog	summary list of all scans in the databroker
bluesky_snapshot	Take a snapshot of a list of EPICS PVs and record it in the databroker.

1.2 bluesky_snapshot

Take a snapshot of a list of EPICS PVs and record it in the databroker. Retrieve (and display) that snapshot later using `APS_BlueSky_tools.callbacks.SnapshotReport`.

1.2.1 Example - command line

Before using the command-line interface, find out what the *bluesky_snapshot* expects:

```
    $ bluesky_snapshot -h
usage: bluesky_snapshot [-h] [-b BROKER_CONFIG] [-m METADATA_SPEC] [-r] [-v]
                        EPICS_PV [EPICS_PV ...]

record a snapshot of some PVs using Bluesky, ophyd, and databroker
version=0.0.40+26.g323cd35

positional arguments:
  EPICS_PV              EPICS PV name

optional arguments:
  -h, --help            show this help message and exit
  -b BROKER_CONFIG      YAML configuration for databroker, default:
                        mongodb_config
  -m METADATA_SPEC, --metadata METADATA_SPEC
                        additional metadata, enclose in quotes, such as -m
                        "purpose=just tuned, situation=routine"
  -r, --report          suppress snapshot report
  -v, --version         show program's version number and exit
```

The help does not tell you that the default for BROKER_CONFIG is "mongodb_config", a YAML file in one of the default locations where the databroker expects to find it. That's what we have.

We want to snapshot just a couple PVs to show basic use. Here are their current values:

```
    $ caget prj:IOC_CPU_LOAD prj:SYS_CPU_LOAD
prj:IOC_CPU_LOAD               0.900851
prj:SYS_CPU_LOAD               4.50426
```

Here's the snapshot (we'll also set a metadata that says this is an example):

```
    $ bluesky_snapshot prj:IOC_CPU_LOAD prj:SYS_CPU_LOAD -m "purpose=example"

========================================
snapshot: 2019-01-03 17:02:42.922197
========================================

hints: {}
hostname: mint-vm
iso8601: 2019-01-03 17:02:42.922197
login_id: mintadmin@mint-vm
plan_description: archive snapshot of ophyd Signals (usually EPICS PVs)
plan_name: snapshot
plan_type: generator
purpose: example
scan_id: 1
software_versions: {'python': '3.6.6 |Anaconda custom (64-bit)| (default, Jun 28 2018,
↪ 17:14:51) \n[GCC 7.2.0]', 'PyEpics': '3.3.1', 'bluesky': '1.4.1', 'ophyd': '1.3.0',
↪ 'databroker': '0.11.3', 'APS_Bluesky_Tools': '0.0.40+26.g323cd35.dirty'}
time: 1546556562.9231327
uid: 98a86a91-d41e-4965-a048-afa5b982a17c
username: mintadmin

========================== ====== ================ ==================
timestamp                  source name              value
========================== ====== ================ ==================
2019-01-03 17:02:33.930067 PV     prj:IOC_CPU_LOAD 0.8007421685989062
2019-01-03 17:02:33.930069 PV     prj:SYS_CPU_LOAD 10.309472772459404
```

(continues on next page)

(continued from previous page)

```
========================= ====== ================ ==================

exit_status: success
num_events: {'primary': 1}
run_start: 98a86a91-d41e-4965-a048-afa5b982a17c
time: 1546556563.1087885
uid: 026fa69c-45b7-4b45-a3b3-266aadbf7176
```

We have a second IOC (*gov*) that has the same PVs. Let's get them, too.:

```
$ bluesky_snapshot {gov,otz}:{IOC,SYS}_CPU_LOAD -m "purpose=this is an example,␣
↪example=example 2"

========================================
snapshot: 2018-12-20 18:21:53.371995
========================================

example: example 2
hints: {}
iso8601: 2018-12-20 18:21:53.371995
plan_description: archive snapshot of ophyd Signals (usually EPICS PVs)
plan_name: snapshot
plan_type: generator
purpose: this is an example
scan_id: 1
software_versions: {'python': '3.6.2 |Continuum Analytics, Inc.| (default, Jul 20␣
↪2017, 13:51:32) \n[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]', 'PyEpics': '3.3.1',
↪'bluesky': '1.4.1', 'ophyd': '1.3.0', 'databroker': '0.11.3', 'APS_Bluesky_Tools':
↪'0.0.37'}
time: 1545351713.3727024
uid: d5e15ba3-0393-4df3-8217-1b72d82b5cf9

========================== ====== ================ ==================
timestamp                  source name             value
========================== ====== ================ ==================
2018-12-20 18:21:45.488033 PV     gov:IOC_CPU_LOAD 0.22522293126578166
2018-12-20 18:21:45.488035 PV     gov:SYS_CPU_LOAD 10.335244804189122
2018-12-20 18:21:46.910976 PV     otz:IOC_CPU_LOAD 0.10009633509509736
2018-12-20 18:21:46.910973 PV     otz:SYS_CPU_LOAD 11.360899731293234
========================== ====== ================ ==================

exit_status: success
num_events: {'primary': 1}
run_start: d5e15ba3-0393-4df3-8217-1b72d82b5cf9
time: 1545351713.3957422
uid: e033cd99-dcac-4b56-848c-62eede1e4d77
```

You can log text and arrays, too.:

```
$ bluesky_snapshot {gov,otz}:{iso8601,HOSTNAME,{IOC,SYS}_CPU_LOAD} compress \
  -m "purpose=this is an example, example=example 2, look=can snapshot text and␣
↪arrays too, note=no commas in metadata"

========================================
snapshot: 2018-12-20 18:28:28.825551
========================================
```

(continues on next page)

(continued from previous page)

```
example: example 2
hints: {}
iso8601: 2018-12-20 18:28:28.825551
look: can snapshot text and arrays too
note: no commas in metadata
plan_description: archive snapshot of ophyd Signals (usually EPICS PVs)
plan_name: snapshot
plan_type: generator
purpose: this is an example
scan_id: 1
software_versions: {'python': '3.6.2 |Continuum Analytics, Inc.| (default, Jul 20␣
↪2017, 13:51:32) \n[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]', 'PyEpics': '3.3.1',
↪'bluesky': '1.4.1', 'ophyd': '1.3.0', 'databroker': '0.11.3', 'APS_Bluesky_Tools':
↪'0.0.37'}
time: 1545352108.8262713
uid: 7e77708e-9169-45ab-b2b6-4e31534d980a

========================== ====== ================ ==================
timestamp                  source name             value
========================== ====== ================ ==================
2018-12-20 18:24:34.220028 PV     compress         [0.1, 0.2, 0.3]
2018-12-13 14:49:53.121188 PV     gov:HOSTNAME     otz.aps.anl.gov
2018-12-20 18:28:25.093941 PV     gov:IOC_CPU_LOAD 0.1501490058473918
2018-12-20 18:28:25.093943 PV     gov:SYS_CPU_LOAD 10.360270546421546
2018-12-20 18:28:28.817630 PV     gov:iso8601      2018-12-20T18:28:28
2018-12-13 14:49:53.135016 PV     otz:HOSTNAME     otz.aps.anl.gov
2018-12-20 18:28:26.525208 PV     otz:IOC_CPU_LOAD 0.10009727705620367
2018-12-20 18:28:26.525190 PV     otz:SYS_CPU_LOAD 12.937574161543873
2018-12-20 18:28:28.830285 PV     otz:iso8601      2018-12-20T18:28:28
========================== ====== ================ ==================

exit_status: success
num_events: {'primary': 1}
run_start: 7e77708e-9169-45ab-b2b6-4e31534d980a
time: 1545352108.8656788
uid: 0de0ec62-504e-4dbc-ad08-2507d4ed44f9
```

1.2.2 Source code documentation

record a snapshot of some PVs using Bluesky, ophyd, and databroker

USAGE:

```
(base) user@hostname .../pwd $ bluesky_snapshot -h
usage: bluesky_snapshot [-h] [-b BROKER_CONFIG] [-m METADATA_SPEC] [-r] [-v]
                        EPICS_PV [EPICS_PV ...]

record a snapshot of some PVs using Bluesky, ophyd, and databroker
version=0.0.40+26.g323cd35

positional arguments:
  EPICS_PV              EPICS PV name

optional arguments:
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  -b BROKER_CONFIG      YAML configuration for databroker, default:
                        mongodb_config
  -m METADATA_SPEC, --metadata METADATA_SPEC
                        additional metadata, enclose in quotes, such as -m
                        "purpose=just tuned, situation=routine"
  -r, --report          suppress snapshot report
  -v, --version         show program's version number and exit
```

APS_BlueSky_tools.snapshot.**get_args**()
get command line arguments

APS_BlueSky_tools.snapshot.**snapshot_cli**()
given a list of PVs on the command line, snapshot and print report

EXAMPLES:

```
snapshot.py pv1 [more pvs ...]
snapshot.py `cat pvlist.txt`
```

Note that these are equivalent:

```
snapshot.py rpi5bf5:0:humidity rpi5bf5:0:temperature
snapshot.py rpi5bf5:0:{humidity,temperature}
```

1.3 Examples

1.3.1 Example: `plan_catalog()`

The APS_BlueSky_tools package provides an executable that can be used to display a summary of all the scans in the database. The executable wraps the demo function: `plan_catalog()`. It is for demonstration purposes only (since it does not filter the output to any specific subset of scans).

The output is a table, formatted as restructured text, with these columns:

date/time The date and time the scan was started.

short_uid The first characters of the scan's UUID (unique identifier).

id The scan number. (User has control of this and could reset the counter for the next scan.)

plan Name of the plan that initiated this scan.

args Arguments to the plan that initiated this scan.

This is run as a linux console command:

```
aps_bluesky_tools_plan_catalog | tee out.txt
```

The `full output` is almost a thousand lines. Here are the first few lines:

```
================== ========= ==== ======================
date/time           short_uid id   plan                   args
================== ========= ==== ======================
2017-10-26 11:21:28 3fe59011  1    scan                   detectors=['noisy'], num=219, motor=['m1'], start=-1.5, stop=-0.5, per_step=None
2017-10-26 11:21:42 25b4c903  2    scan                   detectors=['noisy'], num=219, motor=['m1'], start=-1.5, stop=-0.5, per_step=None
2017-10-26 11:22:08 3953e8e0  3    scan                   detectors=['noisy'], num=219, motor=['m1'], start=-1.5, stop=-0.5, per_step=None
2017-10-26 11:22:22 f24bf2cc  4    scan                   detectors=['noisy'], num=219, motor=['m1'], start=-1.5, stop=-0.5, per_step=None
2017-10-26 11:22:37 44b751d2  5    scan                   detectors=['noisy'], num=219, motor=['m1'], start=-1.5, stop=-0.5, per_step=None
2017-10-26 11:22:50 4e3741f5  6    scan                   detectors=['noisy'], num=219, motor=['m1'], start=-1.5, stop=-0.5, per_step=None
2017-10-26 11:24:33 a83df5d4  7    scan                   detectors=['synthetic_pseudovoigt'], num=219, motor=['m1'], start=-2, stop=0, per_step=None
```

1.3.2 Example: `specfile_example()`

We'll use a Jupyter notebook to demonstrate the `specfile_example()` that writes one or more scans to a SPEC data file. Follow here: https://github.com/BCDA-APS/APS_BlueSky_tools/blob/master/docs/source/resources/demo_specfile_example.ipynb

1.3.3 Example: `nscan()`

We'll use a Jupyter notebook to demonstrate the `nscan()` plan. An *nscan* is used to scan two or more axes together, such as a θ-2θ diffractometer scan. Follow here: https://github.com/BCDA-APS/APS_BlueSky_tools/blob/master/docs/source/resources/demo_nscan.ipynb

1.3.4 Example: `TuneAxis()`

We'll use a Jupyter notebook to demonstrate the `TuneAxis()` support that provides custom alignment of a signal against an axis. Follow here: https://github.com/BCDA-APS/APS_BlueSky_tools/blob/master/docs/source/resources/demo_tuneaxis.ipynb

1.3.5 Source Code Documentation

demonstrate BlueSky callbacks

plan_catalog(db)	make a table of all scans known in the databroker
specfile_example(headers[, filename])	write one or more headers (scans) to a SPEC data file

APS_BlueSky_tools.examples.**main**()
summary list of all scans in the databroker

aps_bluesky_tools_plan_catalog command-line application

This can be unwieldy if there are many scans in the databroker. Consider it as a demo program rather than for general, long-term use.

APS_BlueSky_tools.examples.**plan_catalog**(*db*)
make a table of all scans known in the databroker

Example:

```
from APS_BlueSky_tools.examples import plan_catalog
plan_catalog(db)
```

APS_BlueSky_tools.examples.**specfile_example**(*headers*, *filename='test_specdata.txt'*)
write one or more headers (scans) to a SPEC data file

1.3.6 Downloads

The jupyter notebook and files related to this section may be downloaded from the following table.

- plan_catalog.txt
- jupyter notebook: demo_nscan
- jupyter notebook: demo_tuneaxis
- jupyter notebook: demo_specfile_example
  - spec1.dat
  - spec2.dat
  - spec3.dat
  - spec_tunes.dat
  - test_specdata.txt

1.4 Callbacks

Callbacks that might be useful at the APS using BlueSky

document_contents_callback(key, doc)	prints document contents – use for diagnosing a document stream
DocumentCollectorCallback()	BlueSky callback to collect *all* documents from most-recent plan

Continued on next page

Table 2 – continued from previous page

SnapshotReport(*args, **kwargs)	show the data from a APS_BlueSky_Tools.plans.snapshot()

FILE WRITER CALLBACK

see SpecWriterCallback()

class APS_BlueSky_tools.callbacks.**DocumentCollectorCallback**
BlueSky callback to collect *all* documents from most-recent plan

Will reset when it receives a *start* document.

EXAMPLE:

```
from APS_BlueSky_tools.callbacks import DocumentCollector
doc_collector = DocumentCollectorCallback()
RE.subscribe(doc_collector.receiver)
...
RE(some_plan())
print(doc_collector.uids)
print(doc_collector.documents["stop"])
```

receiver(*key*, *document*)
keep all documents from recent plan in memory

class APS_BlueSky_tools.callbacks.**SnapshotReport**(**args*, ***kwargs*)
show the data from a APS_BlueSky_Tools.plans.snapshot()

Find most recent snapshot between certain dates:

```
headers = db(plan_name="snapshot", since="2018-12-15", until="2018-12-21")
h = list(headers)[0]        # pick the first one, it's the most recent
APS_BlueSky_Tools.callbacks.SnapshotReport().print_report(h)
```

Use as callback to a snapshot plan:

```
RE(
    APS_BlueSky_Tools.plans.snapshot(ophyd_objects_list),
    APS_BlueSky_Tools.callbacks.SnapshotReport()
)
```

descriptor(*doc*)

special case: the data is both in the descriptor AND the event docs due to the way our plan created it

print_report(*header*)
simplify the job of writing our custom data table

method: play the entire document stream through this callback

APS_BlueSky_tools.callbacks.**document_contents_callback**(*key*, *doc*)
prints document contents – use for diagnosing a document stream

1.5 Devices

(ophyd) Devices that might be useful at the APS using BlueSky

APS GENERAL SUPPORT

ApsMachineParametersDevice(*args, **kwargs)	common operational parameters of the APS of general interest
ApsPssShutter(*args, **kwargs)	APS PSS shutter
ApsPssShutterWithStatus(prefix, state_pv, ...)	APS PSS shutter with separate status PV
SimulatedApsPssShutterWithStatus(*args, **kwargs)	Simulated APS PSS shutter

AREA DETECTOR SUPPORT

AD_setup_FrameType(prefix[, scheme])	configure so frames are identified & handled by type (dark, white, or image)
AD_warmed_up(detector)	Has area detector pushed an NDarray to the HDF5 plugin? True or False
AD_EpicsHdf5FileName(*args, **kwargs)	custom class to define image file name from EPICS

DETECTOR / SCALER SUPPORT

use_EPICS_scaler_channels(scaler)	configure scaler for only the channels with names assigned in EPICS

MOTORS, POSITIONERS, AXES, ...

AxisTunerException	Exception during execution of *AxisTunerBase* subclass
AxisTunerMixin(*args, **kwargs)	Mixin class to provide tuning capabilities for an axis
EpicsDescriptionMixin(*args, **kwargs)	add a record's description field to a Device, such as EpicsMotor
EpicsMotorDialMixin(*args, **kwargs)	add motor record's dial coordinate fields to Device
EpicsMotorLimitsMixin(*args, **kwargs)	add motor record HLM & LLM fields & compatibility get_lim() and set_lim()
EpicsMotorRawMixin(*args, **kwargs)	add motor record's raw coordinate fields to Device
EpicsMotorServoMixin(*args, **kwargs)	add motor record's servo loop controls to Device
EpicsMotorShutter(*args, **kwargs)	a shutter, implemented with an EPICS motor moved between two positions
EpicsOnOffShutter(*args, **kwargs)	a shutter, implemented with an EPICS PV moved between two positions

SHUTTERS

ApsPssShutter(*args, **kwargs)	APS PSS shutter
ApsPssShutterWithStatus(prefix, state_pv, ...)	APS PSS shutter with separate status PV
EpicsMotorShutter(*args, **kwargs)	a shutter, implemented with an EPICS motor moved between two positions
EpicsOnOffShutter(*args, **kwargs)	a shutter, implemented with an EPICS PV moved between two positions

synApps records

busyRecord(*args, **kwargs)	
sscanRecord(*args, **kwargs)	EPICS synApps sscan record: used as $(P):scan(N)

Continued on next page

Table 8 – continued from previous page

sscanDevice(*args, **kwargs)	synApps XXX IOC setup of sscan records: $(P):scan$(N)
swaitRecord(*args, **kwargs)	synApps swait record: used as $(P):userCalc$(N)
swait_setup_random_number(swait, **kw)	setup swait record to generate random numbers
swait_setup_gaussian(swait, motor[, center, ...])	setup swait for noisy Gaussian
swait_setup_lorentzian(swait, motor[, ...])	setup swait record for noisy Lorentzian
swait_setup_incrementer(swait[, scan, limit])	setup swait record as an incrementer
userCalcsDevice(*args, **kwargs)	synApps XXX IOC setup of userCalcs: $(P):userCalc$(N)

OTHER SUPPORT

DualPf4FilterBox(*args, **kwargs)	Dual Xia PF4 filter boxes using support from synApps (using Al, Ti foils)
EpicsDescriptionMixin(*args, **kwargs)	add a record's description field to a Device, such as EpicsMotor
ProcedureRegistry(*args, **kwargs)	Procedure Registry: run a blocking function in a thread

Internal routines

ApsOperatorMessagesDevice(*args, **kwargs)	general messages from the APS main control room
DeviceMixinBase(*args, **kwargs)	Base class for APS_Bluesky_tools Device mixin classes

class APS_BlueSky_tools.devices.**AD_EpicsHdf5FileName**(**args*, ***kwargs*)

custom class to define image file name from EPICS

> **Caution:** *Caveat emptor* applies here. You assume expertise!

Replace standard Bluesky algorithm where file names are defined as UUID strings, virtually guaranteeing that no existing images files will ever be overwritten.

Also, this method decouples the data files from the databroker, which needs the files to be named by UUID.

make_filename()	overrides default behavior: Get info from EPICS HDF5 plugin.
generate_datum(key, timestamp, datum_kwargs)	Generate a uid and cache it with its key for later insertion.
get_frames_per_point()	overrides default behavior
stage()	overrides default behavior

To allow users to control the file **name**, we override the `make_filename()` method here and we need to override some intervening classes.

To allow users to control the file **number**, we override the `stage()` method here and triple-comment out that line, and bring in sections from the methods we are replacing here.

The image file name is set in *FileStoreBase.make_filename()* from *ophyd.areadetector.filestore_mixins*. This is called (during device staging) from *FileStoreBase.stage()*

EXAMPLE:

To use this custom class, we need to connect it to some intervening structure. Here are the steps:

1. override default file naming
2. use to make your custom iterative writer
3. use to make your custom HDF5 plugin
4. use to make your custom AD support

imports:

```
from bluesky import RunEngine, plans as bp
from ophyd.areadetector import SimDetector, SingleTrigger
from ophyd.areadetector import ADComponent, ImagePlugin, SimDetectorCam
from ophyd.areadetector import HDF5Plugin
from ophyd.areadetector.filestore_mixins import FileStoreIterativeWrite
```

override default file naming:

```
from APS_BlueSky_tools.devices import AD_EpicsHdf5FileName
```

make a custom iterative writer:

```
class myHdf5EpicsIterativeWriter(AD_EpicsHdf5FileName, FileStoreIterativeWrite):␣
↪pass
```

make a custom HDF5 plugin:

```
class myHDF5FileNames(HDF5Plugin, myHdf5EpicsIterativeWriter): pass
```

define support for the detector (simulated detector here):

```
class MySimDetector(SingleTrigger, SimDetector):
    '''SimDetector with HDF5 file names specified by EPICS'''

    cam = ADComponent(SimDetectorCam, "cam1:")
    image = ADComponent(ImagePlugin, "image1:")

    hdf1 = ADComponent(
        myHDF5FileNames,
        suffix = "HDF1:",
        root = "/",
        write_path_template = "/",
        )
```

create an instance of the detector:

```
simdet = MySimDetector("13SIM1:", name="simdet")
if hasattr(simdet.hdf1.stage_sigs, "array_counter"):
    # remove this so array counter is not set to zero each staging
    del simdet.hdf1.stage_sigs["array_counter"]
simdet.hdf1.stage_sigs["file_template"] = '%s%s_%3.3d.h5'
```

setup the file names using the EPICS HDF5 plugin:

```
simdet.hdf1.file_path.put("/tmp/simdet_demo/")     # ! ALWAYS end with a "/" !
simdet.hdf1.file_name.put("test")
simdet.hdf1.array_counter.put(0)
```

If you have not already, create a bluesky RunEngine:

```
RE = RunEngine({})
```

take an image:

```
RE(bp.count([simdet]))
```

INTERNAL METHODS

generate_datum(*key*, *timestamp*, *datum_kwargs*)
Generate a uid and cache it with its key for later insertion.

get_frames_per_point()
overrides default behavior

make_filename()
overrides default behavior: Get info from EPICS HDF5 plugin.

stage()
overrides default behavior

Set EPICS items before device is staged, then copy EPICS naming template (and other items) to ophyd after staging.

APS_BlueSky_tools.devices.**AD_setup_FrameType**(*prefix*, *scheme='NeXus'*)
configure so frames are identified & handled by type (dark, white, or image)

PARAMETERS

prefix (str) : EPICS PV prefix of area detector, such as "13SIM1:" scheme (str) : any key in the *AD_FrameType_schemes* dictionary

This routine prepares the EPICS Area Detector to identify frames by image type for handling by clients, such as the HDF5 file writing plugin. With the HDF5 plugin, the *FrameType* PV is added to the NDattributes and then used in the layout file to direct the acquired frame to the chosen dataset. The *FrameType* PV value provides the HDF5 address to be used.

To use a different scheme than the defaults, add a new key to the *AD_FrameType_schemes* dictionary, defining storage values for the fields of the EPICS *mbbo* record that you will be using.

see: https://github.com/BCDA-APS/use_bluesky/blob/master/notebooks/images_darks_flats.ipynb

EXAMPLE:

```
AD_setup_FrameType("2bmbPG3:", scheme="DataExchange")
```

- Call this function *before* creating the ophyd area detector object
- use lower-level PyEpics interface

APS_BlueSky_tools.devices.**AD_warmed_up**(*detector*)
Has area detector pushed an NDarray to the HDF5 plugin? True or False

Works around an observed issue: #598 https://github.com/NSLS-II/ophyd/issues/598#issuecomment-414311372

If detector IOC has just been started and has not yet taken an image with the HDF5 plugin, then a TimeoutError will occur as the HDF5 plugin "Capture" is set to 1 (Start). In such case, first acquire at least one image with the HDF5 plugin enabled.

class APS_BlueSky_tools.devices.**ApsBssUserInfoDevice**(**args*, ***kwargs*)
provide current experiment info from the APS BSS

BSS: Beamtime Scheduling System

EXAMPLE:

```
bss_user_info = ApsBssUserInfoDevice(
                    "9id_bss:",
                    name="bss_user_info")
sd.baseline.append(bss_user_info)
```

class APS_BlueSky_tools.devices.**ApsMachineParametersDevice**(**args*, ***kwargs*)
common operational parameters of the APS of general interest

EXAMPLE:

```
import APS_BlueSky_tools.devices as APS_devices
APS = APS_devices.ApsMachineParametersDevice(name="APS")
aps_current = APS.current

# make sure these values are logged at start and stop of every scan
sd.baseline.append(APS)
# record storage ring current as secondary stream during scans
# name: aps_current_monitor
# db[-1].table("aps_current_monitor")
sd.monitors.append(aps_current)
```

The *sd.baseline* and *sd.monitors* usage relies on this global setup:

from bluesky import SupplementalData sd = SupplementalData() RE.preprocessors.append(sd)

inUserOperations	determine if APS is in User Operations mode (boolean)

inUserOperations
determine if APS is in User Operations mode (boolean)

Use this property to configure ophyd Devices for direct or simulated hardware. See issue #49 (https://github.com/BCDA-APS/APS_BlueSky_tools/issues/49) for details.

EXAMPLE:

```
APS = APS_BlueSky_tools.devices.ApsMachineParametersDevice(name="APS")

if APS.inUserOperations:
    suspend_APS_current = bluesky.suspenders.SuspendFloor(APS.current, 2,␣
↪resume_thresh=10)
    RE.install_suspender(suspend_APS_current)
else:
    # use pseudo shutter controls and no current suspenders
    pass
```

class APS_BlueSky_tools.devices.**ApsOperatorMessagesDevice**(**args*, ***kwargs*)
general messages from the APS main control room

class APS_BlueSky_tools.devices.**ApsPssShutter**(**args*, ***kwargs*)
APS PSS shutter

- APS PSS shutters have separate bit PVs for open and close

- set either bit, the shutter moves, and the bit resets a short time later
- no indication that the shutter has actually moved from the bits (see *ApsPssShutterWithStatus()* for alternative)

EXAMPLE:

```
shutter_a = ApsPssShutter("2bma:A_shutter", name="shutter")

shutter_a.open()
shutter_a.close()

shutter_a.set("open")
shutter_a.set("close")
```

When using the shutter in a plan, be sure to use `yield from`, such as:

```
def in_a_plan(shutter):
    yield from abs_set(shutter, "open", wait=True)
    # do something
    yield from abs_set(shutter, "close", wait=True)

RE(in_a_plan(shutter_a))
```

The strings accepted by *set()* are defined in two lists: *valid_open_values* and *valid_close_values*. These lists are treated (internally to *set()*) as lower case strings.

Example, add “o” & “x” as aliases for “open” & “close”:

shutter_a.valid_open_values.append(“o”) shutter_a.valid_close_values.append(“x”) shutter_a.set(“o”) shutter_a.set(“x”)

close()
request shutter to close, interactive use

open()
request shutter to open, interactive use

set(*value*, ***kwargs*)
request the shutter to open or close, BlueSky plan use

class APS_BlueSky_tools.devices.**ApsPssShutterWithStatus**(*prefix*, *state_pv*, **args*, ***kwargs*)

APS PSS shutter with separate status PV

- APS PSS shutters have separate bit PVs for open and close
- set either bit, the shutter moves, and the bit resets a short time later
- a separate status PV tells if the shutter is open or closed (see *ApsPssShutter()* for alternative)

EXAMPLE:

```
A_shutter = ApsPssShutterWithStatus(
    "2bma:A_shutter",
    "PA:02BM:STA_A_FES_OPEN_PL",
    name="A_shutter")
B_shutter = ApsPssShutterWithStatus(
    "2bma:B_shutter",
    "PA:02BM:STA_B_SBS_OPEN_PL",
    name="B_shutter")
```

(continues on next page)

(continued from previous page)

```
A_shutter.open()
A_shutter.close()

or

%mov A_shutter "open"
%mov A_shutter "close"

or

A_shutter.set("open")       # MUST be "open", not "Open"
A_shutter.set("close")
```

When using the shutter in a plan, be sure to use *yield from*.

def in_a_plan(shutter): yield from abs_set(shutter, "open", wait=True) # do something yield from abs_set(shutter, "close", wait=True)

RE(in_a_plan(A_shutter))

The strings accepted by *set()* are defined in attributes (*open_str* and *close_str*).

close(*timeout=10*)

isClosed

isOpen

open(*timeout=10*)

class APS_BlueSky_tools.devices.**ApsUndulator**(**args*, ***kwargs*)
APS Undulator

EXAMPLE:

```
undulator = ApsUndulator("ID09ds:", name="undulator")
```

class APS_BlueSky_tools.devices.**ApsUndulatorDual**(**args*, ***kwargs*)
APS Undulator with upstream *and* downstream controls

EXAMPLE:

```
undulator = ApsUndulatorDual("ID09", name="undulator")
```

note:: the trailing `:` in the PV prefix should be omitted

exception APS_BlueSky_tools.devices.**AxisTunerException**
Exception during execution of *AxisTunerBase* subclass

class APS_BlueSky_tools.devices.**AxisTunerMixin**(**args*, ***kwargs*)
Mixin class to provide tuning capabilities for an axis

See the *TuneAxis()* example in this jupyter notebook: https://github.com/BCDA-APS/APS_BlueSky_tools/blob/master/docs/source/resources/demo_tuneaxis.ipynb

HOOK METHODS

There are two hook methods (*pre_tune_method()*, and *post_tune_method()*) for callers to add additional plan parts, such as opening or closing shutters, setting detector parameters, or other actions.

Each hook method must accept a single argument: an axis object such as *EpicsMotor* or *SynAxis*, such as:

```
def my_pre_tune_hook(axis):
    yield from bps.mv(shutter, "open")
def my_post_tune_hook(axis):
    yield from bps.mv(shutter, "close")

class TunableSynAxis(AxisTunerMixin, SynAxis): pass

myaxis = TunableSynAxis(name="myaxis")
mydet = SynGauss('mydet', myaxis, 'myaxis', center=0.21, Imax=0.98e5, sigma=0.127)
myaxis.tuner = TuneAxis([mydet], myaxis)
myaxis.pre_tune_method = my_pre_tune_hook
myaxis.post_tune_method = my_post_tune_hook

RE(myaxis.tune())
```

class APS_BlueSky_tools.devices.**DeviceMixinBase**(**args*, ***kwargs*)
Base class for APS_Bluesky_tools Device mixin classes

class APS_BlueSky_tools.devices.**DualPf4FilterBox**(**args*, ***kwargs*)
Dual Xia PF4 filter boxes using support from synApps (using Al, Ti foils)

EXAMPLE:

```
pf4 = DualPf4FilterBox("2bmb:pf4:", name="pf4")
pf4_AlTi = DualPf4FilterBox("9idcRIO:pf4:", name="pf4_AlTi")
```

class APS_BlueSky_tools.devices.**EpicsDescriptionMixin**(**args*, ***kwargs*)
add a record's description field to a Device, such as EpicsMotor

EXAMPLE:

```
from ophyd import EpicsMotor
from APS_BlueSky_tools.devices import EpicsDescriptionMixin

class myEpicsMotor(EpicsDescriptionMixin, EpicsMotor): pass
m1 = myEpicsMotor('xxx:m1', name='m1')
print(m1.desc.value)
```

class APS_BlueSky_tools.devices.**EpicsMotorDialMixin**(**args*, ***kwargs*)
add motor record's dial coordinate fields to Device

EXAMPLE:

```
from ophyd import EpicsMotor
from APS_BlueSky_tools.devices import EpicsMotorDialMixin

class myEpicsMotor(EpicsMotorDialMixin, EpicsMotor): pass
m1 = myEpicsMotor('xxx:m1', name='m1')
print(m1.dial.read())
```

class APS_BlueSky_tools.devices.**EpicsMotorLimitsMixin**(**args*, ***kwargs*)
add motor record HLM & LLM fields & compatibility get_lim() and set_lim()

EXAMPLE:

```
from ophyd import EpicsMotor
from APS_BlueSky_tools.devices import EpicsMotorLimitsMixin
```

(continues on next page)

(continued from previous page)

```
class myEpicsMotor(EpicsMotorLimitsMixin, EpicsMotor): pass
m1 = myEpicsMotor('xxx:m1', name='m1')
lo = m1.get_lim(-1)
hi = m1.get_lim(1)
m1.set_lim(-25, -5)
print(m1.get_lim(-1), m1.get_lim(1))
m1.set_lim(lo, hi)
```

get_lim(*flag*)
Returns the user limit of motor

- flag > 0: returns high limit
- flag < 0: returns low limit
- flag == 0: returns None

Similar with SPEC command

set_lim(*low*, *high*)
Sets the low and high limits of motor

- No action taken if motor is moving.
- Low limit is set to lesser of (low, high)
- High limit is set to greater of (low, high)

Similar with SPEC command

class APS_BlueSky_tools.devices.**EpicsMotorRawMixin**(**args*, ***kwargs*)
add motor record's raw coordinate fields to Device

EXAMPLE:

```
from ophyd import EpicsMotor
from APS_BlueSky_tools.devices import EpicsMotorRawMixin

class myEpicsMotor(EpicsMotorRawMixin, EpicsMotor): pass
m1 = myEpicsMotor('xxx:m1', name='m1')
print(m1.raw.read())
```

class APS_BlueSky_tools.devices.**EpicsMotorServoMixin**(**args*, ***kwargs*)
add motor record's servo loop controls to Device

EXAMPLE:

```
from ophyd import EpicsMotor
from APS_BlueSky_tools.devices import EpicsMotorServoMixin

class myEpicsMotor(EpicsMotorServoMixin, EpicsMotor): pass
m1 = myEpicsMotor('xxx:m1', name='m1')
print(m1.servo.read())
```

class APS_BlueSky_tools.devices.**EpicsMotorShutter**(**args*, ***kwargs*)
a shutter, implemented with an EPICS motor moved between two positions

EXAMPLE:

```
tomo_shutter = EpicsMotorShutter("2bma:m23", name="tomo_shutter")
tomo_shutter.closed_position = 1.0      # default
tomo_shutter.open_position = 0.0        # default
tomo_shutter.open()
tomo_shutter.close()

# or, when used in a plan
def planA():
    yield from abs_set(tomo_shutter, "open", group="O")
    yield from wait("O")
    yield from abs_set(tomo_shutter, "close", group="X")
    yield from wait("X")
def planA():
    yield from abs_set(tomo_shutter, "open", wait=True)
    yield from abs_set(tomo_shutter, "close", wait=True)
def planA():
    yield from mv(tomo_shutter, "open")
    yield from mv(tomo_shutter, "close")
```

close()
move motor to BEAM BLOCKED position, interactive use

isClosed

isOpen

open()
move motor to BEAM NOT BLOCKED position, interactive use

set(*value*, *, *timeout=None*, *settle_time=None*)
set() is like *put()*, but used in BlueSky plans

PARAMETERS

value : “open” or “close”

timeout [float, optional] Maximum time to wait. Note that set_and_wait does not support an infinite timeout.

settle_time: float, optional Delay after the set() has completed to indicate completion to the caller

RETURNS

status : DeviceStatus

class APS_BlueSky_tools.devices.**EpicsOnOffShutter**(**args*, ***kwargs*)
a shutter, implemented with an EPICS PV moved between two positions

Use for a shutter controlled by a single PV which takes a value for the close command and a different value for the open command. The current position is determined by comparing the value of the control with the expected open and close values.

EXAMPLE:

```
bit_shutter = EpicsOnOffShutter("2bma:bit1", name="bit_shutter")
bit_shutter.closed_position = 0      # default
bit_shutter.open_position = 1        # default
bit_shutter.open()
bit_shutter.close()

# or, when used in a plan
```

(continues on next page)

(continued from previous page)

```
def planA():
    yield from mv(bit_shutter, "open")
    yield from mv(bit_shutter, "close")
```

close()
move control to BEAM BLOCKED position, interactive use

isClosed

isOpen

open()
move control to BEAM NOT BLOCKED position, interactive use

set(*value*, *, *timeout=None*, *settle_time=None*)
set() is like *put()*, but used in BlueSky plans

PARAMETERS

value : "open" or "close"

timeout [float, optional] Maximum time to wait. Note that set_and_wait does not support an infinite timeout.

settle_time: float, optional Delay after the set() has completed to indicate completion to the caller

RETURNS

status : DeviceStatus

class APS_BlueSky_tools.devices.**ProcedureRegistry**(**args*, ***kwargs*)
Procedure Registry: run a blocking function in a thread

With many instruments, such as USAXS, there are several operating modes to be used, each with its own setup code. This ophyd Device should coordinate those modes so that the setup procedures can be called either as part of a Bluesky plan or from the command line directly. Assumes that users will write functions to setup a particular operation or operating mode. The user-written functions may not be appropriate to use in a plan directly since they might make blocking calls. The ProcedureRegistry will call the function in a thread (which is allowed to make blocking calls) and wait for the thread to complete.

It is assumed that each user-written function will not return until it is complete. .. autosummary:

```
~dir
~add
~remove
~set
~put
```

EXAMPLE:

Given these function definitions:

```
def clearScalerNames():
    for ch in scaler.channels.configuration_attrs:
        if ch.find(".") < 0:
            chan = scaler.channels.__getattribute__(ch)
            chan.chname.put("")

def setMyScalerNames():
    scaler.channels.chan01.chname.put("clock")
```

(continues on next page)

(continued from previous page)

```
    scaler.channels.chan02.chname.put("I0")
    scaler.channels.chan03.chname.put("detector")
```

create a registry and add the two functions (default name is the function name):

use_mode = ProcedureRegistry(name="ProcedureRegistry") use_mode.add(clearScalerNames) use_mode.add(setMyScalerNames)

and then use this registry in a plan, such as this:

```
def myPlan():
    yield from bps.mv(use_mode, "setMyScalerNames")
    yield from bps.sleep(5)
    yield from bps.mv(use_mode, "clearScalerNames")
```

add(*procedure*, *proc_name=None*)
add procedure to registry

dir
tuple of procedure names

put(*value*)
replaces ophyd Device default put() behavior

remove(*procedure*)
remove procedure from registry

set(*proc_name*)
run procedure in a thread, return once it is complete

proc_name (str) : name of registered procedure to be run

class APS_BlueSky_tools.devices.**SimulatedApsPssShutterWithStatus**(**args*, ***kwargs*)

Simulated APS PSS shutter

EXAMPLE:

```
sim = SimulatedApsPssShutterWithStatus(name="sim")
```

close(*timeout=10*)
request the shutter to close

get_response_time()
simulated response time for PSS status

isClosed
is the shutter closed?

isOpen
is the shutter open?

open(*timeout=10*)
request the shutter to open

set(*value*, ***kwargs*)
set the shutter to "close" or "open"

APS_BlueSky_tools.devices.**use_EPICS_scaler_channels**(*scaler*)
configure scaler for only the channels with names assigned in EPICS

1.6 File Writers

BlueSky callback that writes SPEC data files

SpecWriterCallback([filename, auto_write])	collect data from BlueSky RunEngine documents to write as SPEC data

EXAMPLE : the *specfile_example()* writes one or more scans to a SPEC data file using a jupyter notebook.

EXAMPLE : use as BlueSky callback:

```
from APS_BlueSky_tools.filewriters import SpecWriterCallback
specwriter = SpecWriterCallback()
RE.subscribe(specwriter.receiver)
```

EXAMPLE : use as writer from Databroker:

```
from APS_BlueSky_tools.filewriters import SpecWriterCallback
specwriter = SpecWriterCallback()
for key, doc in db.get_documents(db[-1]):
    specwriter.receiver(key, doc)
print("Look at SPEC data file: "+specwriter.spec_filename)
```

EXAMPLE : use as writer from Databroker with customizations:

```
from APS_BlueSky_tools.filewriters import SpecWriterCallback

# write into file: /tmp/cerium.spec
specwriter = SpecWriterCallback(filename="/tmp/cerium.spec")
for key, doc in db.get_documents(db[-1]):
    specwriter.receiver(key, doc)

# write into file: /tmp/barium.dat
specwriter.newfile("/tmp/barium.dat")
for key, doc in db.get_documents(db["b46b63d4"]):
    specwriter.receiver(key, doc)
```

class APS_BlueSky_tools.filewriters.**SpecWriterCallback**(*filename=None*, *auto_write=True*)

collect data from BlueSky RunEngine documents to write as SPEC data

This gathers data from all documents and appends scan to the file when the *stop* document is received.

Parameters

filename [string, optional] Local, relative or absolute name of SPEC data file to be used. If *filename=None*, defaults to format of YYYmmdd-HHMMSS.dat derived from the current system time.

auto_write [boolean, optional] If True (default), *write_scan()* is called when *stop* document is received. If False, the caller is responsible for calling *write_scan()* before the next *start* document is received.

User Interface methods

receiver(key, document)	BlueSky callback: receive all documents for handling
newfile([filename, reset_scan_id, RE])	prepare to use a new SPEC data file
usefile(filename)	read from existing SPEC data file

Continued on next page

Table 14 – continued from previous page

make_default_filename()	generate a file name to be used as default
clear()	reset all scan data defaults
prepare_scan_contents()	format the scan for a SPEC data file
write_scan()	write the most recent (completed) scan to the file

Internal methods

write_header()	write the header section of a SPEC data file
start(doc)	handle *start* documents
descriptor(doc)	handle *descriptor* documents
event(doc)	handle *event* documents
bulk_events(doc)	handle *bulk_events* documents
datum(doc)	handle *datum* documents
resource(doc)	handle *resource* documents
stop(doc)	handle *stop* documents

bulk_events(*doc*)
handle *bulk_events* documents

clear()
reset all scan data defaults

datum(*doc*)
handle *datum* documents

descriptor(*doc*)
handle *descriptor* documents

prepare for primary scan data, ignore any other data stream

event(*doc*)
handle *event* documents

make_default_filename()
generate a file name to be used as default

newfile(*filename=None*, *reset_scan_id=False*, *RE=None*)
prepare to use a new SPEC data file

but don't create it until we have data

prepare_scan_contents()
format the scan for a SPEC data file

Returns [str] a list of lines to append to the data file

receiver(*key*, *document*)
BlueSky callback: receive all documents for handling

resource(*doc*)
handle *resource* documents

start(*doc*)
handle *start* documents

stop(*doc*)
handle *stop* documents

usefile (*filename*)
read from existing SPEC data file

write_header()
write the header section of a SPEC data file

write_scan()
write the most recent (completed) scan to the file

- creates file if not existing
- writes header if needed
- appends scan data

note: does nothing if there are no lines to be written

Example output from `SpecWriterCallback()`:

```
#F test_specdata.txt
#E 1510948301
#D Fri Nov 17 13:51:41 2017
#C BlueSky  user = mintadmin  host = mint-vm

#S 233  scan(detectors=['synthetic_pseudovoigt'], num=20, motor=['m1'], start=-1.65,␣
↪stop=-1.25, per_step=None)
#D Fri Nov 17 11:58:56 2017
#C Fri Nov 17 11:58:56 2017.  plan_type = generator
#C Fri Nov 17 11:58:56 2017.  uid = ddb81ac5-f3ee-4219-b047-c1196d08a5c1
#MD beamline_id = developer__YOUR_BEAMLINE_HERE
#MD login_id = mintadmin@mint-vm
#MD motors = ['m1']
#MD num_intervals = 19
#MD num_points = 20
#MD pid = 7133
#MD plan_pattern = linspace
#MD plan_pattern_args = {'start': -1.65, 'stop': -1.25, 'num': 20}
#MD plan_pattern_module = numpy
#MD proposal_id = None
#N 20
#L m1  m1_user_setpoint  Epoch_float  Epoch  synthetic_pseudovoigt
-1.6500000000000001 -1.65 8.27465009689331 8 2155.6249784809206
-1.6288 -1.6289473684210525 8.46523666381836 8 2629.5229081466964
-1.608 -1.6078947368421053 8.665581226348877 9 3277.4074328018964
-1.5868 -1.5868421052631578 8.865738153457642 9 4246.145049452576
-1.5656 -1.5657894736842104 9.066259145736694 9 5825.186516381953
-1.5448000000000002 -1.5447368421052632 9.266754627227783 9 8803.414029867528
-1.5236 -1.5236842105263158 9.467074871063232 9 15501.419687691103
-1.5028000000000001 -1.5026315789473683 9.667330741882324 10 29570.38936784884
-1.4816 -1.4815789473684209 9.867793798446655 10 55562.3437459487
-1.4604000000000001 -1.4605263157894737 10.067811012268066 10 89519.64275090238
-1.4396 -1.4394736842105262 10.268356084823608 10 97008.97190269837
-1.4184 -1.418421052631579 10.470621824264526 10 65917.29757650592
-1.3972 -1.3973684210526316 10.669955730438232 11 36203.46726798266
-1.3764 -1.3763157894736842 10.870310306549072 11 18897.64061096024
-1.3552 -1.3552631578947367 11.070487976074219 11 10316.223844200193
-1.3344 -1.3342105263157895 11.271018743515015 11 6540.179615556269
-1.3132000000000001 -1.313157894736842 11.4724280834198 11 4643.555421314616
-1.292 -1.2921052631578946 11.673305034637451 12 3533.8582404216445
-1.2712 -1.2710526315789474 11.874176025390625 12 2809.1872596809008
```

(continues on next page)

(continued from previous page)

```
-1.25 -1.25 12.074703216552734 12 2285.9226305883626
#C Fri Nov 17 11:59:08 2017.  num_events_primary = 20
#C Fri Nov 17 11:59:08 2017.  time = 2017-11-17 11:59:08.324011
#C Fri Nov 17 11:59:08 2017.  exit_status = success
```

1.7 Plans

Plans that might be useful at the APS when using BlueSky

nscan(detectors, *motor_sets[, num, ...])	Scan over `n` variables moved together, each in equally spaced steps.
ProcedureRegistry(*args, **kwargs)	Procedure Registry
run_blocker_in_plan(blocker, *args[, ...])	plan: run blocking function `blocker_(*args, **kwargs)` from a Bluesky plan
run_in_thread(func)	(decorator) run `func` in thread
snapshot(obj_list[, stream, md])	bluesky plan: record current values of list of ophyd signals
TuneAxis(signals, axis[, signal_name])	tune an axis with a signal
tune_axes(axes)	BlueSky plan to tune a list of axes in sequence

class APS_BlueSky_tools.plans.**ProcedureRegistry**(**args*, ***kwargs*)

Procedure Registry

> **Caution:** This Device may be relocated or removed entirely in future releases. Its use is complicated and could lead to instability.

With many instruments, such as USAXS, there are several operating modes to be used, each with its own setup code. This ophyd Device should coordinate those modes so that the setup procedures can be called either as part of a Bluesky plan or from the command line directly.

Assumes that users will write functions to setup a particular operation or operating mode. The user-written functions may not be appropriate to use in a plan directly since they might make blocking calls. The ProcedureRegistry will call the function in a thread (which is allowed to make blocking calls) and wait for the thread to complete.

It is assumed that each user-written function will not return until it is complete.

dir	tuple of procedure names
add(procedure[, proc_name])	add procedure to registry
remove(procedure)	remove procedure from registry
set(proc_name)	run procedure in a thread, return once it is complete
put(value)	replaces ophyd Device default put() behavior

EXAMPLE:

```
use_mode = ProcedureRegistry(name="use_mode")

def clearScalerNames():
    for ch in scaler.channels.configuration_attrs:
```

(continues on next page)

(continued from previous page)

```
        if ch.find(".") < 0:
            chan = scaler.channels.__getattribute__(ch)
            chan.chname.put("")

def setMyScalerNames():
    scaler.channels.chan01.chname.put("clock")
    scaler.channels.chan02.chname.put("I0")
    scaler.channels.chan03.chname.put("detector")

def useMyScalerNames(): # Bluesky plan
    yield from bps.mv(
        m1, 5,
        use_mode, "clear",
    )
    yield from bps.mv(
        m1, 0,
        use_mode, "set",
    )

def demo():
    print(1)
    m1.move(5)
    print(2)
    time.sleep(2)
    print(3)
    m1.move(0)
    print(4)


use_mode.add(demo)
use_mode.add(clearScalerNames, "clear")
use_mode.add(setMyScalerNames, "set")
# use_mode.set("demo")
# use_mode.set("clear")
# RE(useMyScalerNames())
```

add(*procedure*, *proc_name=None*)
add procedure to registry

dir
tuple of procedure names

put(*value*)
replaces ophyd Device default put() behavior

remove(*procedure*)
remove procedure from registry

set(*proc_name*)
run procedure in a thread, return once it is complete

proc_name (str) : name of registered procedure to be run

class APS_BlueSky_tools.plans.**TuneAxis**(*signals*, *axis*, *signal_name=None*)
tune an axis with a signal

This class provides a tuning object so that a Device or other entity may gain its own tuning process, keeping track of the particulars needed to tune this device again. For example, one could add a tuner to a motor stage:

```
motor = EpicsMotor("xxx:motor", "motor")
motor.tuner = TuneAxis([det], motor)
```

Then the `motor` could be tuned individually:

```
RE(motor.tuner.tune(md={"activity": "tuning"}))
```

or the *tune()* could be part of a plan with other steps.

Example:

```
tuner = TuneAxis([det], axis)
live_table = LiveTable(["axis", "det"])
RE(tuner.multi_pass_tune(width=2, num=9), live_table)
RE(tuner.tune(width=0.05, num=9), live_table)
```

Also see the jupyter notebook referenced here: *Example: TuneAxis()*.

tune([width, num, md])	BlueSky plan to execute one pass through the current scan range
multi_pass_tune([width, step_factor, num, ...])	BlueSky plan for tuning this axis with this signal
peak_detected()	returns True if a peak was detected, otherwise False

multi_pass_tune(*width=None*, *step_factor=None*, *num=None*, *pass_max=None*, *snake=None*, *md=None*)

BlueSky plan for tuning this axis with this signal

Execute multiple passes to refine the centroid determination. Each subsequent pass will reduce the width of scan by `step_factor`. If `snake=True` then the scan direction will reverse with each subsequent pass.

PARAMETERS

width [float] width of the tuning scan in the units of `self.axis` Default value in `self.width` (initially 1)

num [int] number of steps Default value in `self.num` (initially 10)

step_factor [float] This reduces the width of the next tuning scan by the given factor. Default value in `self.step_factor` (initially 4)

pass_max [int] Maximum number of passes to be executed (avoids runaway scans when a centroid is not found). Default value in `self.pass_max` (initially 10)

snake [bool] If `True`, reverse scan direction on next pass. Default value in `self.snake` (initially True)

md [dict, optional] metadata

peak_detected()

returns True if a peak was detected, otherwise False

The default algorithm identifies a peak when the maximum value is four times the minimum value. Change this routine by subclassing *TuneAxis* and override *peak_detected()*.

tune(*width=None*, *num=None*, *md=None*)

BlueSky plan to execute one pass through the current scan range

Scan self.axis centered about current position from `-width/2` to `+width/2` with `num` observations. If a peak was detected (default check is that max >= 4*min), then set `self.tune_ok = True`.

PARAMETERS

width [float] width of the tuning scan in the units of `self.axis` Default value in `self.width` (initially 1)

num [int] number of steps Default value in `self.num` (initially 10)

md [dict, optional] metadata

`APS_BlueSky_tools.plans.`**`nscan`**(*detectors*, **motor_sets*, *num=11*, *per_step=None*, *md=None*)

Scan over `n` variables moved together, each in equally spaced steps.

PARAMETERS

detectors [list] list of 'readable' objects

motor_sets [list] sequence of one or more groups of: motor, start, finish

motor [object] any 'settable' object (motor, temp controller, etc.)

start [float] starting position of motor

finish [float] ending position of motor

num [int] number of steps (default = 11)

per_step [callable, optional] hook for customizing action of inner loop (messages per step) Expected signature: `f(detectors, step_cache, pos_cache)`

md [dict, optional] metadata

See the *nscan()* example in a Jupyter notebook: https://github.com/BCDA-APS/APS_BlueSky_tools/blob/master/docs/source/resources/demo_nscan.ipynb

`APS_BlueSky_tools.plans.`**`run_blocker_in_plan`**(*blocker*, **args*, *_poll_s_=0.01*, *_timeout_s_=None*, ***kwargs*)

plan: run blocking function `blocker_(*args, **kwargs)` from a Bluesky plan

PARAMETERS

blocker [func] function object to be called in a Bluesky plan

_poll_s_ [float] sleep interval in loop while waiting for completion (default: 0.01)

_timeout_s_ [float] maximum time for completion (default: *None* which means no timeout)

Example: use `time.sleep` as blocking function:

```
RE(run_blocker_in_plan(time.sleep, 2.14))
```

Example: in a plan, use `time.sleep` as blocking function:

```
def my_sleep(t=1.0):
    yield from run_blocker_in_plan(time.sleep, t)

RE(my_sleep())
```

`APS_BlueSky_tools.plans.`**`run_in_thread`**(*func*)

(decorator) run `func` in thread

USAGE:

```
@run_in_thread
def progress_reporting():
    logger.debug("progress_reporting is starting")
```

(continues on next page)

(continued from previous page)

```
    # ...

#...
progress_reporting()    # runs in separate thread
#...
```

APS_BlueSky_tools.plans.**snapshot**(*obj_list*, *stream='primary'*, *md=None*)

bluesky plan: record current values of list of ophyd signals

PARAMETERS

obj_list [list] list of ophyd Signal or EpicsSignal objects

stream [str] document stream, default: "primary"

md [dict] metadata

APS_BlueSky_tools.plans.**tune_axes**(*axes*)

BlueSky plan to tune a list of axes in sequence

EXAMPLE

Sequentially, tune a list of preconfigured axes:

```
RE(tune_axes([mr, m2r, ar, a2r])
```

1.8 Signals

(ophyd) Signals that might be useful at the APS using Bluesky

SynPseudoVoigt(name, motor, motor_field[, ...])	Evaluate a point on a pseudo-Voigt based on the value of a motor.

class APS_BlueSky_tools.signals.**SynPseudoVoigt**(*name*, *motor*, *motor_field*, *center=0*, *eta=0.5*, *scale=1*, *sigma=1*, *bkg=0*, *noise=None*, *noise_multiplier=1*, ***kwargs*)

Evaluate a point on a pseudo-Voigt based on the value of a motor.

Provides a signal to be measured. Acts like a detector.

See https://en.wikipedia.org/wiki/Voigt_profile

PARAMETERS

name [str] name of detector signal

motor [*Mover*] The independent coordinate

motor_field [str] name of *Mover* field

center [float, optional] location of maximum value, default=0

eta [float, optional] 0 <= eta < 1.0: Lorentzian fraction, default=0.5

scale [float, optional] scale >= 1 : scale factor, default=1

sigma [float, optional] sigma > 0 : width, default=1

bkg [float, optional] bkg >= 0 : constant background, default=0

noise [{'poisson', 'uniform', None}] Add noise to the result.

noise_multiplier [float] Only relevant for 'uniform' noise. Multiply the random amount of noise by 'noise_multiplier'

EXAMPLE

```
from APS_BlueSky_tools.signals import SynPseudoVoigt
motor = Mover('motor', {'motor': lambda x: x}, {'x': 0})
det = SynPseudoVoigt('det', motor, 'motor',
    center=0, eta=0.5, scale=1, sigma=1, bkg=0)
```

EXAMPLE

```
import numpy as np
from APS_BlueSky_tools.signals import SynPseudoVoigt
synthetic_pseudovoigt = SynPseudoVoigt(
    'synthetic_pseudovoigt', m1, 'm1',
    center=-1.5 + 0.5*np.random.uniform(),
    eta=0.2 + 0.5*np.random.uniform(),
    sigma=0.001 + 0.05*np.random.uniform(),
    scale=1e5,
    bkg=0.01*np.random.uniform())

#  RE(bp.scan([synthetic_pseudovoigt], m1, -2, 0, 219))
```

1.9 Suspenders

(bluesky) custom support for pausing a running plan

SuspendWhenChanged(signal, *[, ...])	Bluesky suspender

class APS_BlueSky_tools.suspenders.**SuspendWhenChanged**(*signal*, *, *expected_value=None*, *allow_resume=False*, *sleep=0*, *pre_plan=None*, *post_plan=None*, *tripped_message=''*, ***kwargs*)

Bluesky suspender

Suspend when the monitored value deviates from the expected. Only resume if allowed AND when monitored equals expected. Default expected value is current value when object is created.

USAGE:

```
# pause if this value changes in our session
# note: this suspender is designed to require Bluesky restart if value changes
suspend_instrument_in_use = SuspendWhenChanged(instrument_in_use)
RE.install_suspender(suspend_instrument_in_use)
```

1.10 Utilities

Various utilities

connect_pvlist(pvlist[, wait, timeout, ...])	given a list of EPICS PV names, return a dictionary of EpicsSignal objects
EmailNotifications([sender])	send email notifications when requested
ExcelDatabaseFileBase()	base class: read-only support for Excel files, treat them like databases
ExcelDatabaseFileGeneric(filename[, labels_row])	Generic (read-only) handling of Excel spreadsheet-as-database
ipython_profile_name()	return the name of the current ipython profile or *None*
print_snapshot_list(db, **search_criteria)	print (stdout) a list of all snapshots in the databroker
text_encode(source)	encode `source` using the default codepoint
to_unicode_or_bust(obj[, encoding])	from: http://farmdev.com/talks/unicode/
unix_cmd(command_list)	run a UNIX command, returns (stdout, stderr)

class APS_BlueSky_tools.utils.**EmailNotifications**(*sender=None*)
send email notifications when requested

use default OS mail utility (so no credentials needed)

send(*subject*, *message*)
send `message` to all addresses

class APS_BlueSky_tools.utils.**ExcelDatabaseFileBase**
base class: read-only support for Excel files, treat them like databases

EXAMPLE

Show how to read an Excel file where one of the columns contains a unique key. This allows for random access to each row of data by use of the *key*.

```
class ExhibitorsDB(ExcelDatabaseFileBase):
    '''
    content for Exhibitors, vendors, and Sponsors from the Excel file
    '''
    EXCEL_FILE = os.path.join("resources", "exhibitors.xlsx")
    LABELS_ROW = 2

    def handle_single_entry(self, entry):
        '''any special handling for a row from the Excel file'''
        pass

    def handleExcelRowEntry(self, entry):
        '''identify the unique key for this entry (row of the Excel file)'''
        key = entry["Name"]
        self.db[key] = entry
```

class APS_BlueSky_tools.utils.**ExcelDatabaseFileGeneric**(*filename*, *labels_row=3*)
Generic (read-only) handling of Excel spreadsheet-as-database

Table labels are given on Excel row `N`, `self.labels_row = N-1`

handleExcelRowEntry(*entry*)
use row number as the unique key

APS_BlueSky_tools.utils.**connect_pvlist**(*pvlist*, *wait=True*, *timeout=2*, *poll_interval=0.1*)
given a list of EPICS PV names, return a dictionary of EpicsSignal objects

PARAMETERS

pvlist [list(str)] list of EPICS PV names

wait [bool] should wait for EpicsSignal objects to connect, default: True

timeout [float] maximum time to wait for PV connections, seconds, default: 2.0

poll_interval [float] time to sleep between checks for PV connections, seconds, default: 0.1

APS_BlueSky_tools.utils.**ipython_profile_name**()
return the name of the current ipython profile or *None*

Example (add to default RunEngine metadata):

```
RE.md['ipython_profile'] = str(ipython_profile_name())
print("using profile: " + RE.md['ipython_profile'])
```

APS_BlueSky_tools.utils.**print_snapshot_list**(*db*, ***search_criteria*)
print (stdout) a list of all snapshots in the databroker

USAGE:

```
print_snapshot_list(db, )
print_snapshot_list(db, purpose="this is an example")
print_snapshot_list(db, since="2018-12-21", until="2019")
```

EXAMPLE:

```
In [16]: from APS_BlueSky_tools.utils import print_snapshot_list
    ...: from APS_BlueSky_tools.callbacks import SnapshotReport
    ...: print_snapshot_list(db, since="2018-12-21", until="2019")
    ...:
= ======== ========================== ==================
# uid      date/time                  purpose
= ======== ========================== ==================
0 d7831dae 2018-12-21 11:39:52.956904 this is an example
1 5049029d 2018-12-21 11:39:30.062463 this is an example
2 588e0149 2018-12-21 11:38:43.153055 this is an example
= ======== ========================== ==================

In [17]: SnapshotReport().print_report(db["5049029d"])

========================================
snapshot: 2018-12-21 11:39:30.062463
========================================

example: example 2
hints: {}
iso8601: 2018-12-21 11:39:30.062463
look: can snapshot text and arrays too
note: no commas in metadata
plan_description: archive snapshot of ophyd Signals (usually EPICS PVs)
plan_name: snapshot
plan_type: generator
purpose: this is an example
scan_id: 1
software_versions: {
    'python':
        '''3.6.2 |Continuum Analytics, Inc.| (default, Jul 20 2017, 13:51:32)
        [GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]''',
    'PyEpics': '3.3.1',
    'bluesky': '1.4.1',
    'ophyd': '1.3.0',
```

(continues on next page)

(continued from previous page)

```
    'databroker': '0.11.3',
    'APS_Bluesky_Tools': '0.0.38'
    }
time: 1545413970.063167
uid: 5049029d-075c-453c-96d2-55431273852b

========================== ====== ================ ==================
timestamp                  source name            value
========================== ====== ================ ==================
2018-12-20 18:24:34.220028 PV     compress         [0.1, 0.2, 0.3]
2018-12-13 14:49:53.121188 PV     gov:HOSTNAME     otz.aps.anl.gov
2018-12-21 11:39:24.268148 PV     gov:IOC_CPU_LOAD 0.22522317161410768
2018-12-21 11:39:24.268151 PV     gov:SYS_CPU_LOAD 9.109026666525944
2018-12-21 11:39:30.017643 PV     gov:iso8601      2018-12-21T11:39:30
2018-12-13 14:49:53.135016 PV     otz:HOSTNAME     otz.aps.anl.gov
2018-12-21 11:39:27.705304 PV     otz:IOC_CPU_LOAD 0.1251210270549924
2018-12-21 11:39:27.705301 PV     otz:SYS_CPU_LOAD 11.611234438304471
2018-12-21 11:39:30.030321 PV     otz:iso8601      2018-12-21T11:39:30
========================== ====== ================ ==================

exit_status: success
num_events: {'primary': 1}
run_start: 5049029d-075c-453c-96d2-55431273852b
time: 1545413970.102147
uid: 6c1b2100-1ef6-404d-943e-405da9ada882
```

APS_BlueSky_tools.utils.**text_encode**(*source*)
encode source using the default codepoint

APS_BlueSky_tools.utils.**to_unicode_or_bust**(*obj*, *encoding='utf-8'*)
from: http://farmdev.com/talks/unicode/

APS_BlueSky_tools.utils.**unix_cmd**(*command_list*)
run a UNIX command, returns (stdout, stderr)

1.11 synApps busy record

see the synApps busy module suppport: https://github.com/epics-modules/busy

Ophyd support for the EPICS busy record

Public Structures

busyRecord(*args, **kwargs)

class APS_BlueSky_tools.synApps_ophyd.busy.**busyRecord**(**args*, ***kwargs*)

1.12 synApps sscan record

see the synApps sscan module suppport: https://github.com/epics-modules/sscan

Ophyd support for the EPICS synApps sscan record

EXAMPLE

import APS_BlueSky_tools.synApps_ophyd scans = APS_BlueSky_tools.synApps_ophyd.sscanDevice("xxx:", name="scans")

Public Structures

sscanRecord(*args, **kwargs)	EPICS synApps sscan record: used as $(P):scan(N)
sscanDevice(*args, **kwargs)	synApps XXX IOC setup of sscan records: $(P):scan$(N)

Private Structures

sscanPositioner(prefix, num, **kwargs)	positioner of an EPICS sscan record
sscanDetector(prefix, num, **kwargs)	detector of an EPICS sscan record
sscanTrigger(prefix, num, **kwargs)	detector trigger of an EPICS sscan record

class APS_BlueSky_tools.synApps_ophyd.sscan.**sscanRecord**(**args*, ***kwargs*)
EPICS synApps sscan record: used as $(P):scan(N)

reset()
set all fields to default values

set(*value*, ***kwargs*)
interface to use bps.mv()

class APS_BlueSky_tools.synApps_ophyd.sscan.**sscanDevice**(**args*, ***kwargs*)
synApps XXX IOC setup of sscan records: $(P):scan$(N)

reset()
set all fields to default values

1.13 synApps swait record

The swait record is part of the calc module: https://htmlpreview.github.io/?https://raw.githubusercontent.com/epics-modules/calc/R3-6-1/documentation/swaitRecord.html

see the synApps calc module suppport: https://github.com/epics-modules/calc

Ophyd support for the EPICS synApps swait record

EXAMPLES:;

import APS_BlueSky_tools.synApps_ophyd calcs = APS_BlueSky_tools.synApps_ophyd.userCalcsDevice("xxx:", name="calcs")

calc1 = calcs.calc1 APS_BlueSky_tools.synApps_ophyd.swait_setup_random_number(calc1)

APS_BlueSky_tools.synApps_ophyd.swait_setup_incrementer(calcs.calc2)

calc1.reset()

swaitRecord(*args, **kwargs)	synApps swait record: used as $(P):userCalc$(N)
userCalcsDevice(*args, **kwargs)	synApps XXX IOC setup of userCalcs: $(P):userCalc$(N)
swait_setup_random_number(swait, **kw)	setup swait record to generate random numbers
swait_setup_gaussian(swait, motor[, center, ...])	setup swait for noisy Gaussian

Continued on next page

Table 25 – continued from previous page

swait_setup_lorentzian(swait, motor[, ...])	setup swait record for noisy Lorentzian
swait_setup_incrementer(swait[, scan, limit])	setup swait record as an incrementer

class APS_BlueSky_tools.synApps_ophyd.swait.**swaitRecord**(**args*, ***kwargs*)
synApps swait record: used as $(P):userCalc$(N)

reset()
set all fields to default values

class APS_BlueSky_tools.synApps_ophyd.swait.**userCalcsDevice**(**args*, ***kwargs*)
synApps XXX IOC setup of userCalcs: $(P):userCalc$(N)

reset()
set all fields to default values

APS_BlueSky_tools.synApps_ophyd.swait.**swait_setup_random_number**(*swait*, ***kw*)
setup swait record to generate random numbers

APS_BlueSky_tools.synApps_ophyd.swait.**swait_setup_gaussian**(*swait*, *motor*, *center=0*, *width=1*, *scale=1*, *noise=0.05*)
setup swait for noisy Gaussian

APS_BlueSky_tools.synApps_ophyd.swait.**swait_setup_lorentzian**(*swait*, *motor*, *center=0*, *width=1*, *scale=1*, *noise=0.05*)
setup swait record for noisy Lorentzian

APS_BlueSky_tools.synApps_ophyd.swait.**swait_setup_incrementer**(*swait*, *scan=None*, *limit=100000*)
setup swait record as an incrementer

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

R

S

T

U

W