apstools Documentation

Release 1.6.1+3.gbe718c4.dirty

Pete R. Jemian

2022-05-25 16:48

CONTENTS:

Various Python tools for use with Bluesky at the APS

CHAPTER

ONE

SUMMARY CONTENTS

CHAPTER

TWO

PACKAGE INFORMATION

version	1.6.1+3.gbe718c4.dirty
published	2022-05-25 16:48
copyright	2017-2022, UChicago Argonne, LLC
license	ANL OPEN SOURCE LICENSE (see LICENSE.txt file)
author	Pete R. Jemian <jemian@anl.gov>

2.1 Installation

2.1.1 Installation

The `apstools` package is available for installation by `conda`, `pip`, or from source.

conda

If you are using Anaconda Python and have `conda` installed, install with this command:

```
$ conda install -c aps-anl-tag apstools
```

pip

Released versions of apstools are available on PyPI.

If you have `pip` installed, then you can install:

```
$ pip install apstools
```

source

The latest development versions of apstools can be downloaded from the GitHub repository listed above:

```
$ git clone http://github.com/BCDA-APS/apstools.git
```

To install in the standard Python location:

```
$ cd apstools
$ python setup.py install
```

To install in user's home directory:

```
$ python setup.py install --user
```

To install in an alternate location:

```
$ python setup.py install --prefix=/path/to/installation/dir
```

2.1.2 Required Libraries

The repository's `environment.yml` file lists the additional packages required by `apstools`. Most packages are available as conda packages from https://anaconda.org. The others are available on https://PyPI.python.org. Among the required packages:

- python>=3.7
- bluesky, databroker, ophyd
- h5py
- pandas
- pyEpics
- pyqt=5
- pyRestTable
- qt=5
- spec2nexus
- xlrd

2.2 Applications

2.2.1 bluesky_snapshot

Take a snapshot of a list of EPICS PVs and record it in the databroker. Retrieve (and display) that snapshot later using *apstools.callbacks.snapshot_report.SnapshotReport* or the *bluesky_snapshot_viewer* graphical user interface.

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
```

(continues on next page)

(continued from previous page)

```
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', 'apstools': '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
========================== ====== ================ ==================

exit_status: success
num_events: {'primary': 1}
```

(continues on next page)

(continued from previous page)

```
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', 'apstools': '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', 'apstools': '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
```

2.2.2 bluesky_snapshot_viewer

View a snapshot previously recorded by the *databroker*. This is a GUI program started with the command: *bluesky_snapshot_viewer*

Internally, this tool calls `apstools.callbacks.SnapshotReport` to make the report. There are no command line options or command line help.

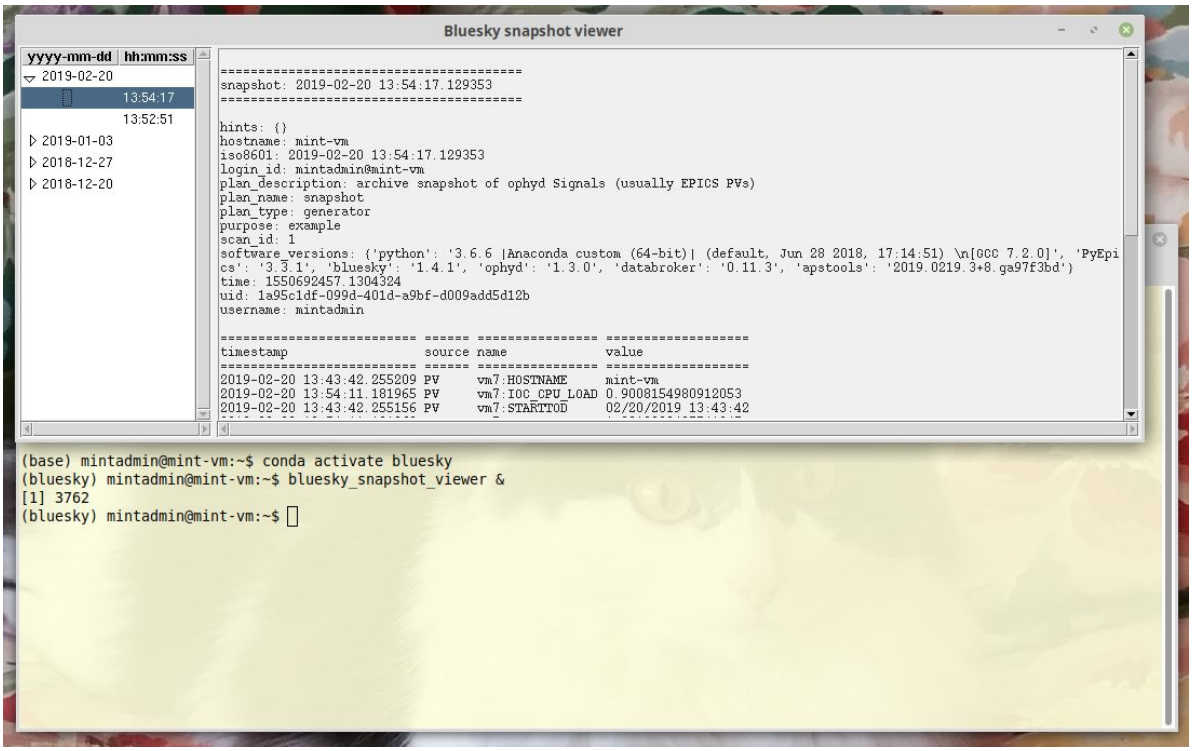


Fig. 1: Screen shot of *bluesky_snapshot_viewer* GUI.

Source code documentation

2.2.3 spec2ophyd

Read SPEC config file and convert to ophyd setup commands.

This is a tool to help migration from SPEC to bluesky. It reads the SPEC configuration file provided on the comand line and converts the lines it recognizes into ophyd commands which create ophyd objects. These commands are printed to *sys.stdout*. The output can be copied into a setup file for ophyd.

Example

SPEC config file

```
# ID @(#)getinfo.c      6.6  01/15/16 CSS
# Device nodes
SDEV_0   = /dev/ttyUSB0 19200 raw
SDEV_1   = /dev/ttyUSB2 19200 raw
#SDEV_2  = /dev/ttyUSB2 19200 raw
VM_EPICS_M1      = 9idcLAX:m58:c0: 8
VM_EPICS_M1      = 9idcLAX:m58:c1: 8
VM_EPICS_M1      = 9idcLAX:m58:c2: 8
VM_EPICS_M1      = 9idcLAX:mxv:c0: 8
VM_EPICS_M1      = 9idcLAX:pi:c0: 4
VM_EPICS_M1      = 9idcLAX:xps:c0: 8
```

(continues on next page)

(continued from previous page)

```
VM_EPICS_M1      = 9idcLAX:aero:c0: 1
VM_EPICS_M1      = 9idcLAX:mxv:c1: 8
VM_EPICS_M1      = 9idcLAX:aero:c1: 1
VM_EPICS_M1      = 9idcLAX:aero:c2: 1
PSE_MAC_MOT      = kohzuE 1
VM_EPICS_M1      = 9ida: 60
VM_EPICS_M1      = 9idcLAX:aero:c3: 1
VM_EPICS_SC      = 9idcLAX:vsc:c0 16
# CAMAC Slot Assignments
#  CA_name_unit = slot [crate_number]
# Motor    cntrl steps sign slew base backl accel nada  flags   mne  name
MOT000 = EPICS_M2:0/2   2000  1  2000  200   50  125    0 0x003       mx  mx
MOT001 = EPICS_M2:0/3   2000  1  2000  200   50  125    0 0x003       my  my
MOT002 = EPICS_M2:1/1   2000  1  2000  200   50  125    0 0x003      msx  msx
MOT003 = EPICS_M2:1/2   2000  1  2000  200   50  125    0 0x003      msy  msy
MOT004 = EPICS_M2:1/3   2000  1  2000  200   50  125    0 0x003      art  ART50-100
MOT005 = EPICS_M2:2/5   2000  1  2000  200   50  125    0 0x003  uslvcen  uslitvercen
MOT006 = EPICS_M2:2/6   2000  1  2000  200   50  125    0 0x003  uslhcen  uslithorcen
MOT007 = EPICS_M2:3/3   2000  1  2000  200   50  125    0 0x003   gslout  GSlit_outb
MOTPAR:read_mode = 7
MOT008 = EPICS_M2:3/4   2000  1  2000  200   50  125    0 0x003   gslinb  GSlit_inb
MOTPAR:read_mode = 7
MOT009 = EPICS_M2:3/5   2000  1  2000  200   50  125    0 0x003   gsltop  GSlit_top
MOTPAR:read_mode = 7
MOT010 = EPICS_M2:3/6   2000  1  2000  200   50  125    0 0x003   gslbot  GSlit_bot
MOTPAR:read_mode = 7
MOT011 = MAC_MOT:0/0   2000  1  2000  200   50  125    0 0x003        en  en
MOTPAR:read_mode = 7
MOT012 = EPICS_M2:10/43   2000  1  2000  200   50  125    0 0x003    InbMS  MonoSl_inb
# Counter   ctrl unit chan scale flags    mne  name
CNT000 = EPICS_SC  0  0 10000000 0x001      sec  seconds
CNT001 = EPICS_SC  0  1      1 0x002       I0  I0
CNT002 = EPICS_SC  0  2      1 0x000      I00  I00
CNT003 = EPICS_SC  0  3      1 0x000     upd2  photodiode
CNT004 = EPICS_SC  0  4      1 0x000      trd  TR_diode
CNT005 = EPICS_SC  0  5      1 0x000     I000  I000
```

command line

Translate the SPEC config file in the present directory:

```
spec2ophyd ./config
```

output

```
mx = EpicsMotor('9idcLAX:m58:c0:m2', name='mx', labels=('motor',))
my = EpicsMotor('9idcLAX:m58:c0:m3', name='my', labels=('motor',))
msx = EpicsMotor('9idcLAX:m58:c1:m1', name='msx', labels=('motor',))
msy = EpicsMotor('9idcLAX:m58:c1:m2', name='msy', labels=('motor',))
art = EpicsMotor('9idcLAX:m58:c1:m3', name='art', labels=('motor',))  # ART50-100
uslvcen = EpicsMotor('9idcLAX:m58:c2:m5', name='uslvcen', labels=('motor',))  #␣
↪uslitvercen
uslhcen = EpicsMotor('9idcLAX:m58:c2:m6', name='uslhcen', labels=('motor',))  #␣
↪uslithorcen
gslout = EpicsMotor('9idcLAX:mxv:c0:m3', name='gslout', labels=('motor',))  # GSlit_outb
↪# read_mode=7
gslinb = EpicsMotor('9idcLAX:mxv:c0:m4', name='gslinb', labels=('motor',))  # GSlit_inb
↪# read_mode=7
gsltop = EpicsMotor('9idcLAX:mxv:c0:m5', name='gsltop', labels=('motor',))  # GSlit_top
↪# read_mode=7
gslbot = EpicsMotor('9idcLAX:mxv:c0:m6', name='gslbot', labels=('motor',))  # GSlit_bot
↪# read_mode=7
# Macro Motor: SpecMotor(mne='en', config_line='11', macro_prefix='kohzuE') # read_mode=7
InbMS = EpicsMotor('9ida:m43', name='InbMS', labels=('motor',))  # MonoSl_inb
c0 = ScalerCH('9idcLAX:vsc:c0', name='c0', labels=('detectors',))
# counter: sec = SpecCounter(mne='sec', config_line='0', name='seconds', unit='0', chan=
↪'0', pvname=9idcLAX:vsc:c0.S1)
# counter: I0 = SpecCounter(mne='I0', config_line='1', unit='0', chan='1',␣
↪pvname=9idcLAX:vsc:c0.S2)
# counter: I00 = SpecCounter(mne='I00', config_line='2', unit='0', chan='2',␣
↪pvname=9idcLAX:vsc:c0.S3)
# counter: upd2 = SpecCounter(mne='upd2', config_line='3', name='photodiode', unit='0',␣
↪chan='3', pvname=9idcLAX:vsc:c0.S4)
# counter: trd = SpecCounter(mne='trd', config_line='4', name='TR_diode', unit='0', chan=
↪'4', pvname=9idcLAX:vsc:c0.S5)
# counter: I000 = SpecCounter(mne='I000', config_line='5', unit='0', chan='5',␣
↪pvname=9idcLAX:vsc:c0.S6)
```

Cautions

- *spec2ophyd* is a work-in-progress.
- *spec2ophyd* does not rely on any libraries of *apstools*
- It is not necessarily robust
- It is not packaged or installed with the apstools.
- It is only available from the source code repository.
- It may be refactored or removed at any time.
- Check the *apstools* Change History for more updates (https://github.com/BCDA-APS/apstools/blob/master/CHANGES.rst)

There are the applications provided by apstools:

application	purpose
bluesky_snapshot	Take a snapshot of a list of EPICS PVs and record it in the databroker.
spec2ophyd	read SPEC config file and convert to ophyd setup commands

2.3 Examples

2.3.1 Example: Pilatus EPICS Area Detector

In this example, we'll show how to create an ophyd object that operates our Pilatus camera as a detector. We'll show how to use the EPICS Area Detector support to save images into HDF5 data files. In the course of this example, we'll describe how an ophyd Device, such as this area detector support, is configured (a.k.a. *staged*) for data acquisition and also describe how ophyd waits for acquisition to complete using a *status object*.

If you just want the final result, we'll present that first. Or, skip ahead if you want the full *Pilatus Support Code Explained*.

Pilatus Support Code

We built a Python class to describe our Pilatus area detector, then created an ophyd `det` object to talk with our EPICS IOC for the Pilatus. Finally, we configured the `det` object to save HDF files when we count with the detector in Bluesky. When Bluesky is not operating the detector, the controls will revert back to their settings before Bluesky started.

Here is the complete support code:

Listing 1: Pilatus Area Detector support, writing HDF5 image files

```
from ophyd import ADComponent
from ophyd import ImagePlugin
from ophyd import PilatusDetector
from ophyd import SingleTrigger
from ophyd.areadetector.filestore_mixins import FileStoreHDF5IterativeWrite
from ophyd.areadetector.plugins import HDF5Plugin_V34
import os

PILATUS_FILES_ROOT = "/mnt/fileserver/data"
BLUESKY_FILES_ROOT = "/export/raid5/fileshare/data"
TEST_IMAGE_DIR = "test/pilatus/%Y/%m/%d/"

class MyHDF5Plugin(FileStoreHDF5IterativeWrite, HDF5Plugin_V34): ...

class MyPilatusDetector(SingleTrigger, PilatusDetector):
    """Pilatus detector"""

    image = ADComponent(ImagePlugin, "image1:")
    hdf1 = ADComponent(
        MyHDF5Plugin,
        "HDF1:",
        write_path_template=os.path.join(PILATUS_FILES_ROOT, TEST_IMAGE_DIR),
        read_path_template=os.path.join(BLUESKY_FILES_ROOT, TEST_IMAGE_DIR),
    )

```

(continues on next page)

(continued from previous page)

```
det = MyPilatusDetector("Pilatus:", name="det")
det.hdf1.create_directory.put(-5)
det.cam.stage_sigs["image_mode"] = "Single"
det.cam.stage_sigs["num_images"] = 1
det.cam.stage_sigs["acquire_time"] = 0.1
det.cam.stage_sigs["acquire_period"] = 0.105
det.hdf1.stage_sigs["lazy_open"] = 1
det.hdf1.stage_sigs["compression"] = "LZ4"
det.hdf1.stage_sigs["file_template"] = "%s%s_%3.3d.h5"
del det.hdf1.stage_sigs["capture"]
det.hdf1.stage_sigs["capture"] = 1
```

Pilatus Support Code Explained

The EPICS Area Detector[1] has support for many different types of area detector. While the full feature set varies amongst the supported camera types, the general approach to building the necessary device support in ophyd follows a common sequence.

An excellent first step to building the device for an area detector is to first check the list of area detector cameras already supported in ophyd.[2] If your camera is not supported, your next step is to build custom support.[3][4]

On the list of supported cameras, we find the `PilatusDetector`.[5]

Note that ophyd makes a distinction (using the Pilatus here as an example) between `PilatusDetector` and ``PilatusDetectorCam``. We'll clarify that distinction below.

Pay special attention to the *Staging an Ophyd Device* section. Staging is fundamental to use of the detector with data acquisition.

General Structure

Before you can create an ophyd object for your Pilatus detector, you'll need to create an ophyd class that describes the features of the EPICS Area Detector interface you plan to use, such as the camera (*ADPilatus*, in this case) and any plugins such as computations or file writers.

Tip: If your EPICS configuration uses **any** of the plugins, you **must** configure them in ophyd. You can check if you missed any once you have created your detector object by calling its `.missing_plugins()` method. For example, where our example Pilatus IOC uses the `Pilatus:` PV prefix:

```
from ophyd import PilatusDetector
det = PilatusDetector("Pilatus:", name="det")
det.missing_plugins()
```

We expect to see an empty list `[]` as the result of this last command. Otherwise, the list will describe the plugins we'll need to define.

---

[1] https://areadetector.github.io/master/index.html
[2] https://blueskyproject.io/ophyd/area-detector.html#specific-hardware
[3] https://blueskyproject.io/ophyd/area-detector.html#custom-devices
[4] https://blueskyproject.io/ophyd/area-detector.html#custom-plugins-or-cameras
[5] https://blueskyproject.io/ophyd/generated/ophyd.areadetector.detectors.PilatusDetector.html

The general support structure is a Python class that such as this one, that provides for triggering and viewing the image (but not file saving):

Listing 2: General Area Detector support Python code

```
class MyPilatusDetector(SingleTrigger, PilatusDetector):
    """Ophyd support class describing this detector"""

    # cam is already defined by PilatusDetector
    image = ADComponent(ImagePlugin, "image1:")
    # define other plugins here, as needed

det = MyPilatusDetector("Pilatus:", name="det")
```

The Python class is defined where it derives from `PilatusDetector` and adds the `SingleTrigger` capabilities. Note the class we are customizing is always listed last, with additional features (also known as *mixin* classes) given first. That's the way Python wants it.

Then, a Python docstring that describes this structure.

Then, any additional *attributes* (class variable names) and their associated `ADComponent` constructions, such as the Image plugin shown. The second argument to the `ADComponent` comes from the EPICS PV for that plugin, such as `Pilatus:image1:` for the Image plugin.

Finally, we show how the object is created with just the PV prefix for EPICS IOC. The `name="det"` keyword argument is required. It is customary that the name matches the object name for the `MyPilatusDetector()` object.

Staging an Ophyd Device

An important part of data acquisition is configuration of each ophyd *Device*[6] for the acquisition steps. In Bluesky, this is called *staging*[7] and the acquisition is called *triggering*.[8] The complete data acquisition sequence of any ophyd Device proceeds in this order:

step	actions
stage	save the current device settings, then prepare the device for trigger
trigger	tell the device to run its acquisition sequence (returns a status object[9] after starting acquisition)
wait	wait until the status object indicates `done=True`
read	get the data from the device (with timestamps)
unstage	restore the previous device settings (as saved in the stage step)

We won't use the *read* step in this example (but Python steps to read the image are shown below in the *Read the Image into Python* section):

- The EPICS IOC saves the image to a file
- Area detector images, unlike most other data we might handle for data acquisition, consume large resources. We should only load that data into memory at the time we choose, not as a routine practice.
- When using the detector in a Bluesky plan, the RunEngine will get the information *about* the image (name and directory of the file created and the address in the file for the image). This information about the image will be part of the document sent to the databroker.

[6] https://blueskyproject.io/ophyd/device-overview.html?highlight=device
[7] https://blueskyproject.io/ophyd/device-overview.html?highlight=staging#stage-and-unstage
[8] https://blueskyproject.io/ophyd/device-overview.html?highlight=device#trd
[9] https://areadetector.github.io/master/ADPilatus/pilatusDoc.html

The ophyd Area Detector `SingleTrigger` mixin provides the configuration to stage and trigger the *.cam* for acquisition. The staging settings, defined as a Python dictionary, will be applied in the order they have been added to the dictionary (and the restored in reverse order). The dictionary is in each Device's *.stage_sigs* attribute. Without the `SingleTrigger` mixin:

```
>>> from ophyd import PilatusDetector
>>> det = PilatusDetector("Pilatus:", name="det")
>>> det.stage_sigs
OrderedDict()
```

With the `SingleTrigger` mixin:

```
>>> from ophyd import PilatusDetector
>>> from ophyd import SingleTrigger
>>> class MyPilatusDetector(SingleTrigger, PilatusDetector): ...
>>> det = MyPilatusDetector("Pilatus:", name="det")
>>> det.stage_sigs
OrderedDict([('cam.acquire', 0), ('cam.image_mode', 1)])
```

The ophyd documentation has more information about *Staging*.

Build the Support: `MyPilatusDetector`

In most cases, you'll want to describe more than just the camera module that EPICS Area Detector supplies for your detector (such as `ADPilatus`[10]). We want to trigger the camera during data collection, view the image during collection[11], and write the image to a file.[12]

The ophyd `PilatusDetector` class only provides an area detector with support for the *cam* module (the camera controls). Since the additional features we want are not supported by `PilatusDetector`, we'll need to add them.

We'll begin customizing the support in the sections below.

`MyPilatusDetector` class

So, following the general structure shown above, we start our `MyPilatusDetector` class, importing the necessary ophyd packages:

Listing 3: starting our MyPilatusDetector() Python code

```
from ophyd import ImagePlugin
from ophyd import PilatusDetector
from ophyd import SingleTrigger

class MyPilatusDetector(SingleTrigger, PilatusDetector):
    """Ophyd support class describing this detector"""

    image = ADComponent(ImagePlugin, "image1:")
```

We could get the same structure with this class instead:

[10] https://areadetector.github.io/master/ADViewers/ad_viewers.html
[11] https://areadetector.github.io/master/ADCore/NDPluginFile.html
[12] https://blueskyproject.io/ophyd/status.html#status-objects-futures

Listing 4: alternative, equivalent to above

```
from ophyd import AreaDetector
from ophyd import ImagePlugin
from ophyd import PilatusDetectorCam
from ophyd import SingleTrigger

class MyPilatusDetector(SingleTrigger, AreaDetector):
    """Ophyd support class describing this detector"""

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

`PilatusDetectorCam` class

The `ophyd.areadetector.PilatusDetectorCam` class provides an ophyd `Device` interface for the *ADPilatus* camera controls. This support is already included in the `PilatusDetector` class so we do not need to add it (although there is no problem if we add it anyway).

Any useful implementation of an EPICS area detector will support the camera module, which controls the features of the camera and image acquisition. The detector classes defined in `ophyd.areadetector.detectors` all support the cam module appropriate for that detector. They are convenience classes for the repetitive step of adding `cam` support.

HDF5Plugin: Writing images to an HDF5 File

The ophyd `HDF5Plugin` class[13], provides support for the HDF5 File Writing Plugin of EPICS Area Detector.

As the EPICS Area Detector support has changed between various releases, the PVs available have also changed. There are several version of the ophyd `HDF5Plugin` class to track those changes. Pick the highest version of ophyd support that is equal or less than the EPICS Area Detector version used in the IOC. For AD 3.7, the highest available ophyd plugin is `ophyd.areadetector.plugins.HDF5Plugin_V34`:

```
from ophyd.areadetector.plugins import HDF5Plugin_V34
```

We *could* just add this to our custom structure:

```
hdf1 = ADComponent(HDF5Plugin_V34, "HDF:")
```

but we still need an additional mixin to control *where* the files should be written (by the IOC) and read (by Bluesky):

```
from ophyd.areadetector.filestore_mixins import FileStoreHDF5IterativeWrite
```

which means we need to define a custom plugin class to bring these two parts together:

```
class MyHDF5Plugin(FileStoreHDF5IterativeWrite, HDF5Plugin_V34): ...
```

The `FileStoreHDF5IterativeWrite` mixin allows for the file directory paths to be different on the two computers, but expects the files to be available to both the EPICS IOC and the Bluesky session. Thus, the paths may have different first parts, up to a point where they match.

The Pilatus detector is a good example that needs the two paths to be different. It saves files to its own file systems. (If the paths are the same on both computers, it is not necessary to specify the `read_path_template`.) For the Bluesky

[13] https://blueskyproject.io/ophyd/generated/ophyd.areadetector.plugins.HDF5Plugin.html

computer to *see* these files, both computers must share the same filesystem. The exact mount point for the shared filesystem can be different on each. Consider these hypothetical mount points for the same shared `data` directory:

```
PILATUS_FILES_ROOT = "/mnt/fileserver/data"
BLUESKY_FILES_ROOT = "/export/raid5/fileshare/data"
```

To configure the `HDF5Plugin()`, we must configure the `write_path_template` for how the shared filesystem is mounted on the Pilatus computer and the `read_path_template` for how the same shared filesystem is mounted on the Bluesky computer. To set these paths, we modify the above line to be:

```
hdf1 = ADComponent(
    MyHDF5Plugin,
    "HDF1:",
    write_path_template=f"{PILATUS_FILES_ROOT}/",
    read_path_template=f"{BLUESKY_FILES_ROOT}/",
)
```

Tip: EPICS Area Detector file writers require the directory separator at the end of the path and will add one if it is not given. Because ophyd expects the PV to become the value it has set, ophyd will timeout when writing the path if the final directory separator is not provided.

Use Python `os.path.join` to create directory paths!

Instead of constructing a file path as:

```
"/mnt/fileserver/data"
```

you may see:

```
os.path.join("/", "mnt", "fileserver", "data")
```

which builds the path using the separator of the current operating system.

Additionally, we add to the mount point the directory path where our files are to be stored on the shared. Bluesky allows this path to include `datetime` formatting. We use this formatting to add the year (`%Y`), month (`%m`), and day (`%d`) into the path for both `write_path_template` and `read_path_template`:

```
TEST_IMAGE_DIR = "test/pilatus/%Y/%m/%d"
```

With this change, our final change is complete:

```
hdf1 = ADComponent(
    MyHDF5Plugin,
    "HDF1:",
    write_path_template=f"{PILATUS_FILES_ROOT}/{TEST_IMAGE_DIR}/",
    read_path_template=f"{BLUESKY_FILES_ROOT}/{TEST_IMAGE_DIR}/",
)
```

Tip: Later, when it is decided to *change* the directory for the HDF5 image files, be sure to set *both* templates, using the proper mount points for each. Follow the pattern as shown:

```
path = "user_name/experiment/"  # note the trailing slash
det.hdf1.write_path_template.put(os.path.join(PILATUS_FILES_ROOT, path))
det.hdf1.read_path_template.put(os.path.join(BLUESKY_FILES_ROOT, path))
```

Create the Ophyd object

With the custom support for our Pilatus, it is simple to create the ophyd object, once we know the PV prefix used by the EPICS IOC. For this example, we'll assume the prefix is `Pilatus:`:

```
det = MyPilatusDetector("Pilatus:", name="det")
```

Directory for the HDF5 files

Previously, we set the `write_path_template` and `read_path_template` to control the directory where the Pilatus IOC writes the HDF5 files and where Bluesky expects to find them once they are created.

If these additional directories do not exist, we'll get an error when we try to write the HDF5 file. EPICS AD HDF5 plugin will create those directories if the *CreateDirectory* PV (the `create_directory` attribute of the `HDF5Plugin()`) is set to a negative number at least as large as the number of directories to be created. A value of `-5` is usually sufficent. Such as:

```
det.hdf1.create_directory.put(-5)
```

Make this adjustment after creating the `det` object and before acquiring an image.

To change the directory for new HDF5 files:

```
path = "user_name/experiment/"  # note the trailing slash
det.hdf1.write_path_template.put(os.path.join(PILATUS_FILES_ROOT, path))
det.hdf1.read_path_template.put(os.path.join(BLUESKY_FILES_ROOT, path))
```

Staging the Camera Settings

We want to control the number of image frames to be acquired so we stage these `cam` features:

```
>>> det.cam.stage_sigs["image_mode"] = "Single"
>>> det.cam.stage_sigs["num_images"] = 1
```

Also, we want to control the acquire time (actual the time the camera is collecting the image) and period (total time between image frames) for the image:

```
>>> det.cam.stage_sigs["acquire_time"] = 0.1
>>> det.cam.stage_sigs["acquire_period"] = 0.105
```

Staging the HDF5Plugin

We need to configure `hdf1` (the HDF5 plugin) for staging. The defaults are:

```
>>> det.hdf1.stage_sigs
OrderedDict([('enable', 1),
             ('blocking_callbacks', 'Yes'),
             ('parent.cam.array_callbacks', 1),
             ('auto_increment', 'Yes'),
             ('array_counter', 0),
             ('auto_save', 'Yes'),
             ('num_capture', 0),
             ('file_template', '%s%s_%6.6d.h5'),
             ('file_write_mode', 'Stream'),
             ('capture', 1)])
```

These settings enable the HDF5 writer and will pause the next acquisition until the HDF5 file is written. They will increment the file numbering and will automatically save the file once the image is captured. By default, ophyd will choose a file name based on a random `uuid`.[14] It is possible to change this naming style but those steps are beyond this example.

We want to enable the `LazyOpen` feature[15] (so we do not have to acquire an image into the HDF5 plugin before our first data acquisition):

```
>>> det.hdf1.stage_sigs["lazy_open"] = 1
```

and we want to add LZ4 compression:

```
>>> det.hdf1.stage_sigs["compression"] = "LZ4"
```

The `LazyOpen` setting *must* happen before the plugin is set to `Capture`, so we must delete that and then add it as the last action:

```
>>> del det.hdf1.stage_sigs["capture"]
>>> det.hdf1.stage_sigs["capture"] = 1
```

We might reduce the number of digits written into the file name (this will change the value in place instead of moving the setting to the end of the actions):

```
>>> det.hdf1.stage_sigs["file_template"] = "%s%s_%3.3d.h5"
```

Acquire and Save an Image

Now that the `det` object is ready for data acquisition, let's acquire an image using the ophyd tools:

```
>>> det.stage()
```

Ack. An upstream problem might appear in response to `det.stage()` as a long exception report, starting with `UnprimedPlugin` and ending with:

```
UnprimedPlugin: The plugin hdf1 on the area detector with name det has not been primed.
```

[14] https://docs.python.org/3/library/uuid.html#uuid.uuid4
[15] `LazyOpen` first appeared in AD 2.2

Until the upstream support in ophyd is corrected to watch for `LazyOpen=1`, you need to *warmup* the plugin (by acquiring an image and pushing it into the HDF plugin):

```
>>> det.warmup()
```

Then, proceed to acquire an image and save it to a file.

```
>>> st = det.trigger()
```

The return result was a Status object. If we check its value before the image is saved to an HDF5 file, the result looks like this:

```
>>> st
ADTriggerStatus(device=det, done=False, success=False)
```

Status objects

Status objects are used when a Device does not complete its action right away. Without Status objects, we would either have to poll the Device to learn if it is finished or we would need to set up an EPICS Channel Access monitor callback function to receive news of changes from the EPICS support. And handle timeouts and failure scenarios. Status objects handle all this routine work for us.

Once the image acquisition is complete, the status object will indicate it is done. We must wait until then by checking it. Or, we can call the `.wait()` method of the status object:

```
>>> st.wait()
```

Once the acquisition is finished and the HDF5 file is written, the `wait()` method will return. We can check its value:

```
>>> st
ADTriggerStatus(device=det, done=True, success=True)
```

Acquisition is complete. Don't forget to `unstage()`:

```
>>> det.unstage()
```

When we use `det` as a detector in a bluesky plan with the `RunEngine`, the `RunEngine` will do all these steps (including the wait for the status object to finish).

We can find the name of the HDF5 that was written (by the IOC):

```
>>> det.hdf1.full_file_name.get()
/mnt/fileserver/data/test/pilatus/2021/01/22/4e26f601-df6d-4848-bf3f_000.h5
```

and we can get a local directory listing of the same file:

```
>>> !ls -lAFgh /export/raid5/fileshare/data/test/pilatus/2021/01/22/4e26f601-df6d-4848-
↪bf3f_000.h5
-rw-r--r-- 1 root 2.2M Jan 22 00:41 /export/raid5/fileshare/data/test/pilatus/2021/01/22/
↪4e26f601-df6d-4848-bf3f_000.h5
```

Note: The file size might be different for your detector.

Read the Image into Python

Our long-term plan is to use `det` for data acquisition with Bluesky and the *databroker* package.[16] Since this example focusses on the ophyd configuration of an area detector, we'll show how to read the image from the HDF5 file. ()

Note: Keep in mind this is not the recommended way to get the image with Bluesky but we show this procedure since we have not used *bluesky* and *databroker* to record the image file details.

Once you have taken an image with `det` and saved it to an HDF5 file, we can read that file and get the image. By default, the EPICS area detector HDF5 File Writer stores the image (using the NeXus schema) at the address `/entry/data/data` in the file.

First, we must get the name of the data file from the IOC:

```
>>> full_name_ioc = det.hdf1.full_file_name.get()
>>> print(f"IOC: {full_name_ioc}")
'/mnt/fileserver/data/test/pilatus/2021/01/22/4e26f601-df6d-4848-bf3f_000.h5'
```

This is the full path as the IOC sees the file system. We can simply remove the IOC path and replace it with the local path:

```
full_name_local = LOCAL_FILES_ROOT + full_name_ioc[len(IOC_FILES_ROOT):]
```

Verify that we have such a file:

```
>>> print(f"local file: {full_name_local}")
'/export/raid5/fileshare/data/test/pilatus/2021/01/22/4e26f601-df6d-4848-bf3f_000.h5'
>>> print(f"exists:{os.path.exists(full_name_local)}")
exists:True
```

Open the file using the *h5py*[17] package:

```
>>> import h5py
>>> root = h5py.File(full_name_local, "r")
```

Read the image:

```
>>> image = root["/entry/data/data"]
```

Show the *shape* of the image:

```
>>> image.shape
(1, 1024, 1024)
```

Close the file:

```
>>> root.close()
```

[16] https://blueskyproject.io/databroker/
[17] https://www.h5py.org/

2.3.2 Example: Perkin-Elmer EPICS Area Detector

In this example, we'll show how to create an ophyd object that operates a Perkin-Elmer Camera as a detector. We'll start with the *Pilatus example*.

The only difference from the Pilatus example is the detector class (`PerkinElmerDetector`), the PV prefix (we'll use `PE1:` here), and possibly the plugin version. For simplicity, we'll assume the same version of EPICS Area Detector (3.7) is used in this example. We'll use a different object name, `det_pe`, for this detector and different directories (where the write and read directories are the same).

Follow the *Pilatus Support Code Explained* to learn more about the choices made below. The process is similar.

Perkin-Elmer Support Code

Here is the Perkin-Elmer support, derived from the Pilatus support (*Example: Pilatus EPICS Area Detector*).

Listing 5: Perkin-Elmer Area Detector support, writing HDF5 image files

```
from ophyd import ADComponent
from ophyd import ImagePlugin
from ophyd import PerkinElmerDetector
from ophyd import SingleTrigger
from ophyd.areadetector.filestore_mixins import FileStoreHDF5IterativeWrite
from ophyd.areadetector.plugins import HDF5Plugin_V34
import os

IMAGE_FILES_ROOT = "/local/data"
TEST_IMAGE_DIR = "test/"

class MyHDF5Plugin(FileStoreHDF5IterativeWrite, HDF5Plugin_V34): ...

class MyPerkinElmerDetector(SingleTrigger, PerkinElmerDetector):
    """Perkin-Elmer detector"""

    image = ADComponent(ImagePlugin, "image1:")
    hdf1 = ADComponent(
        MyHDF5Plugin,
        "HDF1:",
        write_path_template=os.path.join(
            IMAGE_FILES_ROOT, TEST_IMAGE_DIR
        ),
    )

det_pe = MyPerkinElmerDetector("PE1:", name="det_pe")
det_pe.hdf1.create_directory.put(-5)
det_pe.cam.stage_sigs["image_mode"] = "Single"
det_pe.cam.stage_sigs["num_images"] = 1
det_pe.cam.stage_sigs["acquire_time"] = 0.1
det_pe.cam.stage_sigs["acquire_period"] = 0.105
det_pe.hdf1.stage_sigs["lazy_open"] = 1
det_pe.hdf1.stage_sigs["compression"] = "LZ4"
det_pe.hdf1.stage_sigs["file_template"] = "%s%s_%3.3d.h5"
del det_pe.hdf1.stage_sigs["capture"]
det_pe.hdf1.stage_sigs["capture"] = 1
```

2.3.3 Example: the `nscan()` plan

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/apstools/blob/master/docs/source/resources/demo_nscan.ipynb

2.3.4 Example: the `run_command_file()` plan

You can use a text file or an Excel spreadsheet as a multi-sample batch scan tool using the `run_command_file()` plan. This section is divided into these parts.

The Command File

A command file can be written as either a plain text file or a spreadsheet (such as from Microsoft Excel or Libre Office).

Text Command File

For example, given a text command file (named `sample_example.txt`) with content as shown …

```
# example text command file

step_scan 5.07  8.3   0.0  "Water Blank, deionized"
other_scan 5.07  8.3  0.0  "Water Blank, deionized"

# all comment lines will be ignored

this line will not be recognized by execute_command_list()
```

… can be summarized in a bluesky ipython session:

```
In [1]: import apstools.plans
In [2]: apstools.plans.summarize_command_file("sample_example.txt")
Command file: sample_example.xlsx
====== ========== ============================================================
line # action     parameters
====== ========== ============================================================
3      step_scan  5.07, 8.3, 0.0, Water Blank, deionized
4      other_scan 5.07, 8.3, 0.0, Water Blank, deionized
```

(continues on next page)

(continued from previous page)

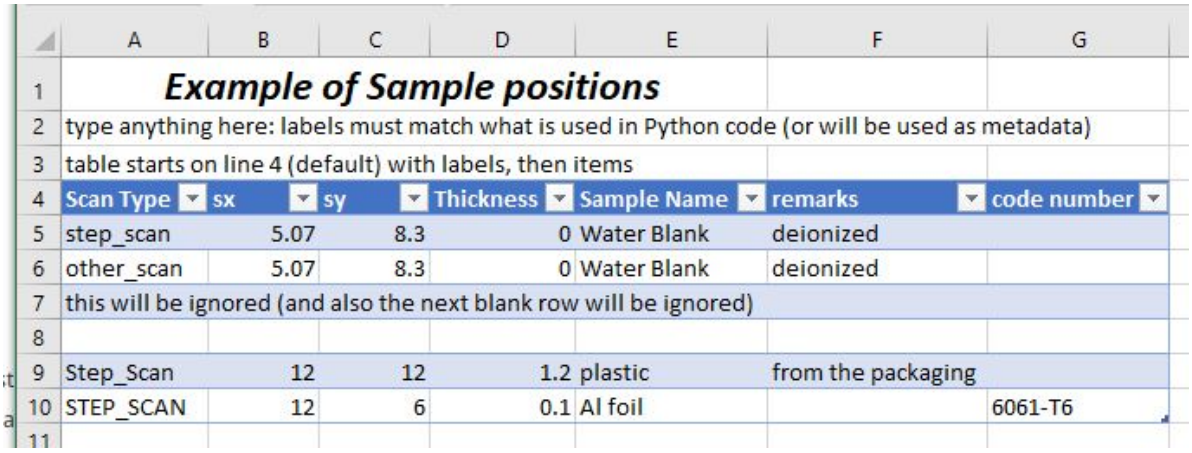
```
8      this       line, will, not, be, recognized, by, execute_command_list()
====== ========== ============================================================
```

Observe which lines (3, 4, and 8) in the text file are part of the summary. The other lines are either comments or blank. The action on line 8 will be passed to `execute_command_list()` which will then report the `this` action it is not handled. (Looks like another type of a comment.) See `parse_text_command_file()` for more details.

Spreadsheet Command File

Follow the example spreadsheet (in the *File Downloads* section) and accompanying Jupyter notebook[1] to write your own `Excel_plan()`.

For example, given a spreadsheet (named `sample_example.xlsx`) with content as shown in the next figure:



	A	B	C	D	E	F	G
1	**Example of Sample positions**						
2	type anything here: labels must match what is used in Python code (or will be used as metadata)						
3	table starts on line 4 (default) with labels, then items						
4	Scan Type	sx	sy	Thickness	Sample Name	remarks	code number
5	step_scan	5.07	8.3	0	Water Blank	deionized	
6	other_scan	5.07	8.3	0	Water Blank	deionized	
7	this will be ignored (and also the next blank row will be ignored)						
8							
9	Step_Scan	12	12	1.2	plastic	from the packaging	
10	STEP_SCAN	12	6	0.1	Al foil		6061-T6
11							

Fig. 2: Image of `sample_example.xlsx` spreadsheet file.

Tip: Place the column labels on the fourth row of the spreadsheet, starting in the first column. The actions start on the next row. The first blank row indicates the end of the command table within the spreadsheet. Use as many columns as you need, one column per argument.

This spreadsheet can be summarized in a bluesky ipython session:

```
In [1]: import apstools.plans
In [2]: apstools.plans.summarize_command_file("sample_example.xlsx")
Command file: sample_example.xlsx
====== ================================================================
↪==================================
line # action                                                  parameters
====== ================================================================
↪==================================
1      step_scan                                                5.07, 8.3, 0.0,
↪ Water Blank, deionized
2      other_scan                                               5.07, 8.3, 0.0,
↪ Water Blank, deionized
```

(continues on next page)

[1] https://github.com/BCDA-APS/apstools/blob/master/docs/source/resources/excel_scan.ipynb

(continued from previous page)

```
3       this will be ignored (and also the next blank row will be ignored)
====== ================================================================␣
↪====================================
```

Note that lines 9 and 10 in the spreadsheet file are not part of the summary. The spreadsheet handler will stop reading the list of actions at the first blank line. The action described on line 3 will not be handled (since we will not define an action named `this will be ignored (and also the next blank row will be ignored)`). See `parse_Excel_command_file()` for more details.

The *Actions*

To use this example with the `run_command_file()` plan, it is necessary to redefine the `execute_command_list()` plan, we must write a plan to handle every different type of action described in the spreadsheet. For `sample_example.xlsx`, we need to handle the `step_scan` and `other_scan` actions.

Tip: Always write these *actions* as bluesky plans. To test your *actions*, use either `bluesky.simulators.summarize_plan(step_scan())` or `bluesky.simulators.summarize_plan(other_scan())`.

Here are examples of those two actions (and a stub for an additional instrument procedure to make the example more realistic):

```
def step_scan(sx, sy, thickness, title, md={}):
        md["sample_title"] = title                      # log this metadata
        md["sample_thickness"] = thickness      # log this metadata
        yield from bluesky.plan_stubs.mv(
                sample_stage.x, sx,
                sample_stage.y, sy,
        )
        yield from prepare_detector("scintillator")
        yield from bluesky.plans.scan([scaler], motor, 0, 180, 360, md=md)

def other_scan(sx, sy, thickness, title, md={}):
        md["sample_title"] = title                      # log this metadata
        md["sample_thickness"] = thickness      # log this metadata
        yield from bluesky.plan_stubs.mv(
                sample_stage.x, sx,
                sample_stage.y, sy,
        )
        yield from prepare_detector("areadetector")
        yield from bluesky.plans.count([areadetector], md=md)

def prepare_detector(detector_name):
        # for this example, we do nothing
        # we should move the proper detector into position here
        yield from bluesky.plan_stubs.null()
```

Now, we replace `execute_command_list()` with our own definition to include those two actions:

```
def execute_command_list(filename, commands, md={}):
    """our custom execute_command_list"""
    full_filename = os.path.abspath(filename)
```

(continues on next page)

(continued from previous page)

```
    if len(commands) == 0:
        yield from bps.null()
        return

    text = f"Command file: {filename}\n"
    text += str(apstools.utils.command_list_as_table(commands))
    print(text)

    for command in commands:
        action, args, i, raw_command = command
        logger.info(f"file line {i}: {raw_command}")

        _md = {}
        _md["full_filename"] = full_filename
        _md["filename"] = filename
        _md["line_number"] = i
        _md["action"] = action
        _md["parameters"] = args    # args is shorter than parameters, means the same␣
↪thing here

        _md.update(md or {})      # overlay with user-supplied metadata

        action = action.lower()
        if action == "step_scan":
            yield from step_scan(*args, md=_md)
        elif action == "other_scan":
            yield from other_scan(*args, md=_md)

        else:
            logger.info(f"no handling for line {i}: {raw_command}")
```

Register our own `execute_command_list`

Finally, we register our new version of `execute_command_list` (which replaces the default `execute_command_list()`):

```
APS_plans.register_command_handler(execute_command_list)
```

If you wish to verify that your own code has been installed, use this command:

```
print(APS_plans._COMMAND_HANDLER_)
```

If its output contains `apstools.plans.execute_command_list`, you have not registered your own function. However, if the output looks something such as either of these:

```
<function __main__.execute_command_list(filename, commands, md={})>
# or
<function execute_command_list at 0x7f4cf0d616a8>
```

then you have installed your own code.

Testing the command file

As you were developing plans for each of your actions, we showed you how to test that each plan was free of basic syntax and bluesky procedural errors. The `bluesky.simulators.summarize_plan()` function will run through your plan and show you the basic data acquisition steps that will be executed during your plan. Becuase you did not write any blocking code, no hardware should ever be changed by running this plan summary.

To test our command file, run it through the `bluesky.simulators.summarize_plan()` function:

```
bluesky.simulators.summarize_plan(run_command_file("sample_example.txt"))
```

The output will be rather lengthy, if there are no errors. Here are the first few lines:

```
Command file: sample_example.txt
====== ========== ============================================================
line # action     parameters
====== ========== ============================================================
3      step_scan  5.07, 8.3, 0.0, Water Blank, deionized
4      other_scan 5.07, 8.3, 0.0, Water Blank, deionized
8      this       line, will, not, be, recognized, by, execute_command_list()
====== ========== ============================================================

file line 3: step_scan 5.07  8.3   0.0  "Water Blank, deionized"
sample_stage_x -> 5.07
sample_stage_y -> 8.3
================================== Open Run ==================================
m3 -> 0.0
  Read ['scaler', 'm3']
m3 -> 0.5013927576601671
  Read ['scaler', 'm3']
m3 -> 1.0027855153203342
  Read ['scaler', 'm3']
m3 -> 1.5041782729805013
```

and the last few lines:

```
m3 -> 178.99721448467966
  Read ['scaler', 'm3']
m3 -> 179.49860724233983
  Read ['scaler', 'm3']
m3 -> 180.0
  Read ['scaler', 'm3']
================================= Close Run ==================================
file line 4: other_scan 5.07  8.3  0.0  "Water Blank, deionized"
sample_stage_x -> 5.07
sample_stage_y -> 8.3
================================== Open Run ==================================
  Read ['areadetector']
================================= Close Run ==================================
file line 8: this line will not be recognized by execute_command_list()
no handling for line 8: this line will not be recognized by execute_command_list()
```

Running the command file

Prepare the RE

These steps were used to prepare our bluesky ipython session to run the plan:

```
from bluesky import RunEngine
from bluesky.utils import get_history
RE = RunEngine(get_history())

# Import matplotlib and put it in interactive mode.
import matplotlib.pyplot as plt
plt.ion()

# load config from ~/.config/databroker/mongodb_config.yml
from databroker import Broker
db = Broker.named("mongodb_config")

# Subscribe metadatastore to documents.
# If this is removed, data is not saved to metadatastore.
RE.subscribe(db.insert)

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

# Add a progress bar.
from bluesky.utils import ProgressBarManager
pbar_manager = ProgressBarManager()
RE.waiting_hook = pbar_manager

# Register bluesky IPython magics.
from bluesky.magics import BlueskyMagics
get_ipython().register_magics(BlueskyMagics)

# Set up the BestEffortCallback.
from bluesky.callbacks.best_effort import BestEffortCallback
bec = BestEffortCallback()
RE.subscribe(bec)
```

Also, since we are using an EPICS area detector (ADSimDetector) and have just started its IOC, we must process at least one image from the CAM to each of the file writing plugins we'll use (just the HDF1 for us). A procedure has been added to the `ophyd.areadetector` code for this. Here is the command we used for this procedure:

```
areadetector.hdf1.warmup()
```

Run the command file

To run the command file, you need to pass this to an instance of the `bluesky.RunEngine`, defined as `RE` above:

```
RE(apstools.plans.run_command_file("sample_example.txt"))
```

The output will be rather lengthy. Here are the first few lines of the output on my system (your hardware may be different so the exact data columns and values will vary):

```
In [11]: RE(APS_plans.run_command_file("sample_example.txt"))
Command file: sample_example.txt
====== ========== ============================================================
line # action     parameters
====== ========== ============================================================
3      step_scan  5.07, 8.3, 0.0, Water Blank, deionized
4      other_scan 5.07, 8.3, 0.0, Water Blank, deionized
8      this       line, will, not, be, recognized, by, execute_command_list()
====== ========== ============================================================

file line 3: step_scan 5.07  8.3   0.0  "Water Blank, deionized"
Transient Scan ID: 138     Time: 2019-07-04 16:52:15
Persistent Unique Scan ID: 'f40d1c3c-8361-4efc-83f1-072fcd44c1b2'
New stream: 'primary'
+-----------+------------+------------+------------+------------+------------+
|   seq_num |       time |         m3 |      clock |         I0 |      scint |
+-----------+------------+------------+------------+------------+------------+
|         1 | 16:52:34.5 |    0.00000 |    4000000 |          2 |          1 |
|         2 | 16:52:35.2 |    0.50000 |    4000000 |          1 |          2 |
|         3 | 16:52:36.0 |    1.00000 |    4000000 |          1 |          1 |
|         4 | 16:52:36.6 |    1.50000 |    4000000 |          2 |          1 |
|         5 | 16:52:37.3 |    2.01000 |    4000000 |          1 |          2 |
|         6 | 16:52:38.0 |    2.51000 |    4000000 |          2 |          2 |
|         7 | 16:52:38.7 |    3.01000 |    4000000 |          1 |          1 |
|         8 | 16:52:39.3 |    3.51000 |    4000000 |          1 |          2 |
```

and the last few lines:

```
|       352 | 16:56:53.4 |  175.99000 |    4000000 |          3 |          0 |
|       353 | 16:56:54.1 |  176.49000 |    4000000 |          3 |          1 |
|       354 | 16:56:55.0 |  176.99000 |    4000000 |          1 |          3 |
|       355 | 16:56:55.7 |  177.49000 |    4000000 |          2 |          1 |
|       356 | 16:56:56.4 |  177.99000 |    4000000 |          1 |          1 |
|       357 | 16:56:57.1 |  178.50000 |    4000000 |          2 |          1 |
|       358 | 16:56:57.8 |  179.00000 |    4000000 |          2 |          2 |
|       359 | 16:56:58.5 |  179.50000 |    4000000 |          1 |          2 |
|       360 | 16:56:59.2 |  180.00000 |    4000000 |          2 |          1 |
+-----------+------------+------------+------------+------------+------------+
generator scan ['f40d1c3c'] (scan num: 138)



file line 4: other_scan 5.07  8.3  0.0  "Water Blank, deionized"
Transient Scan ID: 139     Time: 2019-07-04 16:56:59
```

(continues on next page)

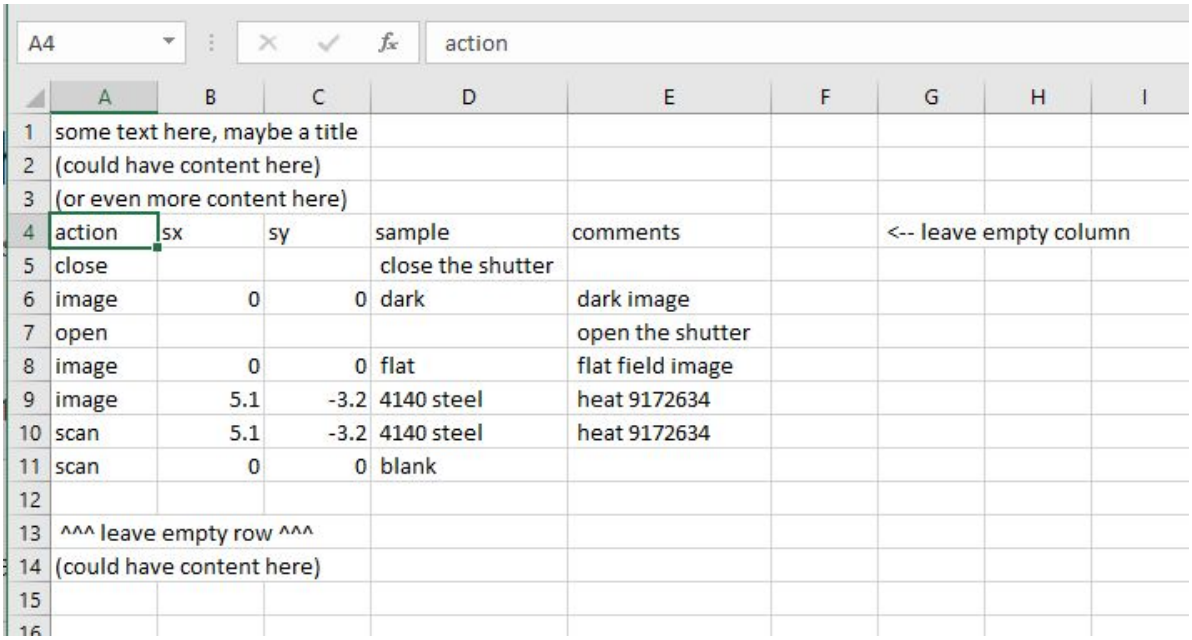
(continued from previous page)

```
Persistent Unique Scan ID: '1f093f56-3413-4e67-8a1b-27e71881b855'
New stream: 'primary'
+-----------+------------+
|   seq_num |       time |
+-----------+------------+
|         1 | 16:56:59.7 |
+-----------+------------+
generator count ['1f093f56'] (scan num: 139)



file line 8: this line will not be recognized by execute_command_list()
no handling for line 8: this line will not be recognized by execute_command_list()
Out[11]:
('f40d1c3c-8361-4efc-83f1-072fcd44c1b2',
 '1f093f56-3413-4e67-8a1b-27e71881b855')

In [12]:
```

Appendix: Other spreadsheet examples

You can use an Excel spreadsheet as a multi-sample batch scan tool.

SIMPLE: Your Excel spreadsheet could be rather simple…



	A	B	C	D	E	F	G	H	I
1	some text here, maybe a title								
2	(could have content here)								
3	(or even more content here)								
4	action	sx	sy	sample	comments		<-- leave empty column		
5	close			close the shutter					
6	image	0	0	dark	dark image				
7	open				open the shutter				
8	image	0	0	flat	flat field image				
9	image	5.1	-3.2	4140 steel	heat 9172634				
10	scan	5.1	-3.2	4140 steel	heat 9172634				
11	scan	0	0	blank					
12									
13	^^^ leave empty row ^^^								
14	(could have content here)								
15									

Fig. 3: Unformatted Excel spreadsheet for batch scans.

See `ExcelDatabaseFileGeneric` for an example bluesky plan that reads from this spreadsheet.

FANCY: … or contain much more information, including formatting.

The idea is that your table will start with column labels in **row 4** of the Excel spreadsheet. One of the columns will be the name of the action (in the example, it is `Scan Type`). The other columns will be parameters or other information.

	A	B	C	D	E	F	G
1	***Example of Sample positions***						
2	type anything here: labels must match what is used in Python code (or will be used as metadata)						
3	table starts on line 4 (default) with labels, then items						
4	Scan Type	sx	sy	Thickness	Sample Name	remarks	code number
5	step_scan	5.07	8.3	0	Water Blank	deionized	
6	other_scan	5.07	8.3	0	Water Blank	deionized	
7	this will be ignored (and also the next blank row will be ignored)						
8							
9	Step_Scan	12	12	1.2	plastic	from the packaging	
10	STEP_SCAN	12	6	0.1	Al foil		6061-T6
11							

Fig. 4: Example Excel spreadsheet for multi-sample batch scans.

Each of the rows under the labels will describe one type of action such as a scan. Basically, whatever you handle in your `Excel_plan()`. Any rows that you do not handle will be reported to the console during execution but will not result in any action. Grow (or shrink) the table as needed.

Note: For now, make sure there is no content in any of the spreadsheet cells outside (either below or to the right) of your table. Such content will trigger a cryptic error about a numpy float that cannot be converted. Instead, put that content in a second spreadsheet page.

You'll need to have an action plan for every different action your spreadsheet will specify. Call these plans from your `Excel_plan()` within an `elif` block, as shown in this example. The example `Excel_plan()` converts the `Scan Type` into lower case for simpler comparisons. Your plan can be different if you choose.

```
if scan_command == "step_scan":
    yield from step_scan(...)
elif scan_command == "energy_scan":
    yield from scan_energy(...)
elif scan_command == "radiograph":
    yield from AcquireImage(...)
else:
    print(f"no handling for table row {i+1}: {row}")
```

The example plan saves all row parameters as metadata to the row's action. This may be useful for diagnostic purposes.

2.3.5 Example: write SPEC data file

We'll use the `demo_specfile_example` Jupyter notebook to demonstrate how to write one or more scans to a SPEC data file.

https://github.com/BCDA-APS/apstools/blob/master/docs/source/resources/demo_specfile_example.ipynb

2.3.6 Example: the `TuneAxis()` class

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/apstools/blob/master/docs/source/resources/demo_tuneaxis.ipynb

2.3.7 File Downloads

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

- jupyter notebook: `demo_excel_scan`
  - `sample_example.xlsx`
- 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`

2.4 API Documentation

2.4.1 Callbacks (includes File Writers)

Callbacks

Document Collector

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

class apstools.callbacks.doc_collector.**DocumentCollectorCallback**
Bluesky callback to collect *all* documents from most-recent plan

Will reset when it receives a *start* document.

EXAMPLE:

```
from apstools.callbacks import DocumentCollectorCallback
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

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

Snapshot Report

SnapshotReport(*args, **kwargs)	Show the data from a `apstools.plans.snapshot()`.

class apstools.callbacks.snapshot_report.**SnapshotReport**(**args: Any*, ***kwargs: Any*)
Show the data from a `apstools.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
apstools.callbacks.SnapshotReport().print_report(h)
```

Use as callback to a snapshot plan:

```
RE(
    apstools.plans.snapshot(ophyd_objects_list),
```

(continues on next page)

(continued from previous page)

```
    apstools.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

File Writers

See the *File Writers* section.

2.4.2 Devices

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

Also consult the Index under the *Ophyd* heading for links to the Devices, Exceptions, Mixins, Signals, and other support items described here.

Categories

APS General Support

ApsCycleDM(*args, **kwargs)	Get the APS cycle name from the APS Data Management system or a local file.
ApsMachineParametersDevice(*args, **kwargs)	Common operational parameters of the APS of general interest.
ApsPssShutter(*args, **kwargs)	APS PSS shutter
ApsPssShutterWithStatus(*args, **kwargs)	APS PSS shutter with separate status PV
SimulatedApsPssShutterWithStatus(*args, **kwargs)	Simulated APS PSS shutter

Area Detector Support

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

Detector & Scaler Support

Struck3820(*args, **kwargs)	Struck/SIS 3820 Multi-Channel Scaler (as used by USAXS)
use_EPICS_scaler_channels(scaler)	configure scaler for only the channels with names assigned in EPICS
SynPseudoVoigt(*args, **kwargs)	Evaluate a point on a pseudo-Voigt based on the value of a motor.

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
EpicsMotorEnableMixin(*args, **kwargs)	mixin providing access to motor enable/disable
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
EpicsMotorResolutionMixin(*args, **kwargs)	Add motor record's resolution fields to motor.
EpicsMotorServoMixin(*args, **kwargs)	add motor record's servo loop controls to Device
PVPositionerSoftDone(*args, **kwargs)	PVPositioner that computes done as a soft signal.
PVPositionerSoftDoneWithStop(*args, **kwargs)	PVPositionerSoftDone with stop() and inposition.
EpicsMotorShutter(*args, **kwargs)	Shutter, implemented with an EPICS motor moved between two positions
EpicsOnOffShutter(*args, **kwargs)	Shutter using a single EPICS PV moved between two positions

Shutters

ApsPssShutter(*args, **kwargs)	APS PSS shutter
ApsPssShutterWithStatus(*args, **kwargs)	APS PSS shutter with separate status PV
EpicsMotorShutter(*args, **kwargs)	Shutter, implemented with an EPICS motor moved between two positions
EpicsOnOffShutter(*args, **kwargs)	Shutter using a single EPICS PV moved between two positions
OneSignalShutter(*args, **kwargs)	Shutter Device using one Signal for open and close.
ShutterBase(*args, **kwargs)	Base class for all shutter Devices.
SimulatedApsPssShutterWithStatus(*args, **kwargs)	Simulated APS PSS shutter

Slits

XiaSlit2D(*args, **kwargs)	EPICS synApps optics xia_slit.db 2D support: inb out bot top ...
Optics2Slit1D(*args, **kwargs)	EPICS synApps optics 2slit.db 1D support: xn, xp, size, center, sync
Optics2Slit2D_HV(*args, **kwargs)	EPICS synApps optics 2slit.db 2D support: h.xn, h.xp, v.xn, v.xp
Optics2Slit2D_InbOutBotTop(*args, **kwargs)	EPICS synApps optics 2slit.db 2D support: inb, out, bot, top
SlitGeometry(width, height, x, y)	Slit size and center as a named tuple

synApps Support

See separate *synApps Support: Records, Databases, ...* section.

Temperature Controllers

Eurotherm2216e(*args, **kwargs)	Eurotherm 2216e Temperature Controller
LakeShore336Device(*args, **kwargs)	LakeShore 336 temperature controller.
LakeShore340Device(*args, **kwargs)	LakeShore 340 temperature controller
Linkam_CI94_Device(*args, **kwargs)	Linkam model CI94 temperature controller
Linkam_T96_Device(*args, **kwargs)	Linkam model T96 temperature controller
PTC10AioChannel(*args, **kwargs)	SRS PTC10 AIO module
PTC10RtdChannel(*args, **kwargs)	SRS PTC10 RTD module channel
PTC10TcChannel(*args, **kwargs)	SRS PTC10 Tc (thermocouple) module channel
PTC10PositionerMixin(*args, **kwargs)	Mixin so SRS PTC10 can be used as a (temperature) positioner.

Other Support

ApsBssUserInfoDevice(*args, **kwargs)	Provide current experiment info from the APS BSS.
Pf4FilterSingle(*args, **kwargs)	XIA PF4 Filter: one set of 4 filters (A).
Pf4FilterDual(*args, **kwargs)	XIA PF4 Filter: two sets of 4 filters (A, B).
Pf4FilterTriple(*args, **kwargs)	XIA PF4 Filter: three sets of 4 filters (A, B, C).
Pf4FilterBank(*args, **kwargs)	A single module of XIA PF4 filters (4-blades).
Pf4FilterCommon(*args, **kwargs)	XIA PF4 filters - common support.
DualPf4FilterBox(*args, **kwargs)	LEGACY (use Pf4FilterDual now): Dual (Al, Ti) Xia PF4 filter boxes
EpicsDescriptionMixin(*args, **kwargs)	add a record's description field to a Device, such as EpicsMotor
KohzuSeqCtl_Monochromator(*args, **kwargs)	synApps Kohzu double-crystal monochromator sequence control program
SRS570_PreAmplifier(*args, **kwargs)	Ophyd support for Stanford Research Systems 570 preamplifier from synApps.

continues on next page

Table 10 – continued from previous page

Struck3820(*args, **kwargs)	Struck/SIS 3820 Multi-Channel Scaler (as used by USAXS)

Internal Routines

ApsOperatorMessagesDevice(*args, **kwargs)	General messages from the APS main control room.
TrackingSignal(*args, **kwargs)	Non-EPICS signal for use when coordinating Device actions.
DeviceMixinBase(*args, **kwargs)	Base class for apstools Device mixin classes

All Submodules

APS User Proposal and ESAF Information

ApsBssUserInfoDevice(*args, **kwargs)	Provide current experiment info from the APS BSS.

class apstools.devices.aps_bss_user.**ApsBssUserInfoDevice**(**args: Any*, ***kwargs: Any*)

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)
```

NOTE: There is info provided by the APS proposal & ESAF systems.

Area Detector Support

AD_setup_FrameType(prefix[, scheme])	configure so frames are identified & handled by type (dark, white, or image)
AD_plugin_primed(plugin)	Has area detector pushed an NDarray to the file writer plugin? True or False
AD_prime_plugin(detector, plugin)	Prime this area detector's file writer plugin.
AD_prime_plugin2(plugin)	Prime this area detector's file writer plugin.
AD_EpicsHdf5FileName(*args, **kwargs)	custom class to define image file name from EPICS
AD_EpicsJpegFileName(*args, **kwargs)	custom class to define image file name from EPICS

class apstools.devices.area_detector_support.**AD_EpicsHdf5FileName**(**args: Any*, ***kwargs: Any*)

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 apstools.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'''
```

(continues on next page)

(continued from previous page)

```
    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.

class apstools.devices.area_detector_support.**AD_EpicsJpegFileName**(**args: Any*, ***kwargs: Any*)
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 JPEG 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 Set EPICS items before device is staged, then copy EPICS naming template (and other items) to ophyd after staging.

Patterned on `apstools.devices.AD_EpicsHdf5FileName()`. (Follow that documentation from this point.)

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 JPEG plugin.

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

apstools.devices.area_detector_support.**AD_plugin_primed**(*plugin*)
Has area detector pushed an NDarray to the file writer plugin? True or False

PARAMETERS

plugin *obj* : area detector plugin to be *primed* (such as `detector.hdf1`)

EXAMPLE:

```
AD_plugin_primed(detector.hdf1)
```

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 file writer plugin, then a TimeoutError will occur as the file writer plugin "Capture" is set to 1 (Start). In such case, first acquire at least one image with the file writer plugin enabled.

Also issue in apstools (needs a robust method to detect if primed): https://github.com/BCDA-APS/apstools/issues/464

Since Area Detector release 2.1 (2014-10-14).

The *prime* process is not needed if you select the *LazyOpen* feature with *Stream* mode for the file plugin. *LazyOpen* defers file creation until the first frame arrives in the plugin. This removes the need to initialize the plugin with a dummy frame before starting capture.

apstools.devices.area_detector_support.**AD_prime_plugin**(*detector*, *plugin*)
Prime this area detector's file writer plugin.

PARAMETERS

detector *obj* : area detector (such as `detector`)

plugin *obj* : area detector plugin to be *primed* (such as `detector.hdf1`)

EXAMPLE:

```
AD_prime_plugin(detector, detector.hdf1)
```

apstools.devices.area_detector_support.**AD_prime_plugin2**(*plugin*)
Prime this area detector's file writer plugin.

Collect and push an NDarray to the file writer plugin. Works with all file writer plugins.

Based on `ophyd.areadetector.plugins.HDF5Plugin.warmup()`.

PARAMETERS

plugin *obj* : area detector plugin to be *primed* (such as `detector.hdf1`)

EXAMPLE:

```
AD_prime_plugin2(detector.hdf1)
```

apstools.devices.area_detector_support.**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://nbviewer.jupyter.org/github/BCDA-APS/use_bluesky/blob/main/lessons/sandbox/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 cycles

ApsCycleDM(*args, **kwargs)	Get the APS cycle name from the APS Data Management system or a local file.

class apstools.devices.aps_cycle.**ApsCycleDM**(**args: Any*, ***kwargs: Any*)
Get the APS cycle name from the APS Data Management system or a local file.

This signal is read-only.

APS Machine Parameters

APS machine parameters

ApsMachineParametersDevice(*args, **kwargs)	Common operational parameters of the APS of general interest.
ApsOperatorMessagesDevice(*args, **kwargs)	General messages from the APS main control room.

class apstools.devices.aps_machine.**ApsMachineParametersDevice**(**args: Any*, ***kwargs: Any*)

Common operational parameters of the APS of general interest.

EXAMPLE:

```
import apstools.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)

aps_cycle

alias of *apstools.devices.aps_cycle.ApsCycleDM*

property 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/apstools/issues/49) for details.

EXAMPLE:

```
APS = apstools.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
```

operator_messages

alias of *apstools.devices.aps_machine.ApsOperatorMessagesDevice*

class apstools.devices.aps_machine.**ApsOperatorMessagesDevice**(**args: Any*, ***kwargs: Any*)
General messages from the APS main control room.

APS undulator

ApsUndulator(*args, **kwargs)	APS Undulator
ApsUndulatorDual(*args, **kwargs)	APS Undulator with upstream *and* downstream controls

class apstools.devices.aps_undulator.**ApsUndulator**(**args: Any*, ***kwargs: Any*)
APS Undulator

EXAMPLE:

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

tracking
alias of *apstools.devices.tracking_signal.TrackingSignal*

class apstools.devices.aps_undulator.**ApsUndulatorDual**(**args: Any*, ***kwargs: Any*)
APS Undulator with upstream *and* downstream controls

EXAMPLE:

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

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

downstream
alias of *apstools.devices.aps_undulator.ApsUndulator*

upstream
alias of *apstools.devices.aps_undulator.ApsUndulator*

Axis Tuner

AxisTunerException	Exception during execution of *AxisTunerBase* subclass
AxisTunerMixin(*args, **kwargs)	Mixin class to provide tuning capabilities for an axis

exception apstools.devices.axis_tuner.**AxisTunerException**
Exception during execution of *AxisTunerBase* subclass

class apstools.devices.axis_tuner.**AxisTunerMixin**(**args: Any*, ***kwargs: Any*)
Mixin class to provide tuning capabilities for an axis

See the *TuneAxis()* example in this jupyter notebook: https://github.com/BCDA-APS/apstools/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

def tune_myaxis():
    yield from myaxis.tune(md={"plan_name": "tune_myaxis"})

RE(tune_myaxis())
```

Mixin to add EPICS .DESC field

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

class apstools.devices.description_mixin.**EpicsDescriptionMixin**(**args: Any*, ***kwargs: Any*)

add a record's description field to a Device, such as EpicsMotor

EXAMPLE:

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

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

more ideas:

```
class TunableSynAxis(AxisTunerMixin, SynAxis):
    '''synthetic axis that can be tuned'''
class TunableEpicsMotor(AxisTunerMixin, EpicsMotor):
    '''EpicsMotor that can be tuned'''
class EpicsMotorWithDescription(EpicsDescriptionMixin, EpicsMotor):
    '''EpicsMotor with description field'''

class EpicsMotorWithMore(
    EpicsDescriptionMixin,
    EpicsMotorLimitsMixin,
    EpicsMotorDialMixin,
    EpicsMotorRawMixin,
    EpicsMotor):
    '''
```

(continues on next page)

(continued from previous page)

```
    EpicsMotor with more fields

    * description (``desc``)
    * soft motor limits (``soft_limit_hi``, ``soft_limit_lo``)
    * dial coordinates (``dial``)
    * raw coordinates (``raw``)
    '''
```

Eurotherm 2216e Temperature Controller

The 2216e is a temperature controller from Eurotherm.

Eurotherm2216e(*args, **kwargs)	Eurotherm 2216e Temperature Controller

According to their website, the [Eurotherm 2216e Temperature Controller](https://www.eurothermcontrollers.com/eurotherm-2216e-series-controller-now-obsolete/) is obsolete. Please see replacement [EPC3016](https://www.eurothermcontrollers.com/eurotherm-epc3016-1-16-din-process-and-temperature-controller/) in our [EPC3000 Series](https://www.eurothermcontrollers.com/epc3000-series).

New in apstools 1.6.0.

class apstools.devices.eurotherm_2216e.**Eurotherm2216e**(**args: Any*, ***kwargs: Any*)

Eurotherm 2216e Temperature Controller

cb_sensor(**args*, ***kwargs*)

units: Convert dC from sensor to C

Kohzu double-crystal monochromator

KohzuSeqCtl_Monochromator(*args, **kwargs)	synApps Kohzu double-crystal monochromator sequence control program

class apstools.devices.kohzu_monochromator.**KohzuSeqCtl_Monochromator**(**args: Any*, ***kwargs: Any*)

synApps Kohzu double-crystal monochromator sequence control program

calibrate_energy(*value*)

Calibrate the monochromator energy.

PARAMETERS

value: float New energy for the current monochromator position.

energy

alias of *apstools.devices.kohzu_monochromator.KohzuSoftPositioner*

theta

alias of *apstools.devices.kohzu_monochromator.KohzuSoftPositioner*

wavelength

alias of *apstools.devices.kohzu_monochromator.KohzuSoftPositioner*

class apstools.devices.kohzu_monochromator.**KohzuSoftPositioner**(**args: Any*, ***kwargs: Any*)

cb_done(**args*, ***kwargs*)
Called when parent's done signal changes (EPICS CA monitor event).

cb_setpoint(**args*, ***kwargs*)
Called when setpoint changes (EPICS CA monitor event).

When the setpoint is changed, force `done=False`. For any move, done must transition to `!= done_value`, then back to `done_value`. Next update will refresh value from parent device.

property inposition
Report (boolean) if positioner is done.

move(**args*, ***kwargs*)
Reposition, with optional wait for completion.

Lakeshore temperature controllers

LakeShore336Device(*args, **kwargs)	LakeShore 336 temperature controller.
LakeShore340Device(*args, **kwargs)	LakeShore 340 temperature controller

class apstools.devices.lakeshore_controllers.**LS340_LoopBase**(**args: Any*, ***kwargs: Any*)
Base settings for both sample and control loops.

class apstools.devices.lakeshore_controllers.**LS340_LoopControl**(**args: Any*, ***kwargs: Any*)
Control specific

class apstools.devices.lakeshore_controllers.**LS340_LoopSample**(**args: Any*, ***kwargs: Any*)
Sample specific

class apstools.devices.lakeshore_controllers.**LakeShore336Device**(**args: Any*, ***kwargs: Any*)
LakeShore 336 temperature controller.

- loop 1: temperature positioner AND heater, PID, & ramp controls
- loop 2: temperature positioner AND heater, PID, & ramp controls
- loop 3: temperature positioner
- loop 4: temperature positioner

loop1
alias of *apstools.devices.lakeshore_controllers.LakeShore336_LoopControl*

loop2
alias of *apstools.devices.lakeshore_controllers.LakeShore336_LoopControl*

loop3
alias of *apstools.devices.lakeshore_controllers.LakeShore336_LoopRO*

loop4
alias of *apstools.devices.lakeshore_controllers.LakeShore336_LoopRO*

serial
alias of *apstools.synApps.asyn.AsynRecord*

class apstools.devices.lakeshore_controllers.**LakeShore336_LoopControl**(**args: Any*, ***kwargs: Any*)
LakeShore 336 temperature controller – with heater control.

The LakeShore 336 accepts up to two heaters.

class apstools.devices.lakeshore_controllers.**LakeShore336_LoopRO**(**args: Any*, ***kwargs: Any*)
LakeShore 336 temperature controller – Read-only loop (no heaters).

class apstools.devices.lakeshore_controllers.**LakeShore340Device**(**args: Any*, ***kwargs: Any*)
LakeShore 340 temperature controller

control
alias of *apstools.devices.lakeshore_controllers.LS340_LoopControl*

sample
alias of *apstools.devices.lakeshore_controllers.LS340_LoopSample*

serial
alias of *apstools.synApps.asyn.AsynRecord*

Linkam temperature controllers

Linkam_CI94_Device(*args, **kwargs)	Linkam model CI94 temperature controller
Linkam_T96_Device(*args, **kwargs)	Linkam model T96 temperature controller

class apstools.devices.linkam_controllers.**Linkam_CI94_Device**(**args: Any*, ***kwargs: Any*)
Linkam model CI94 temperature controller

EXAMPLE:

```
ci94 = Linkam_CI94_Device("IOC:ci94:", name="ci94")
```

temperature
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDoneWithStop*

class apstools.devices.linkam_controllers.**Linkam_T96_Device**(**args: Any*, ***kwargs: Any*)
Linkam model T96 temperature controller

EXAMPLE:

```
tc1 = Linkam_T96("IOC:tc1:", name="tc1")
```

temperature
alias of *apstools.devices.linkam_controllers.T96Temperature*

class apstools.devices.linkam_controllers.**T96Temperature**(**args: Any*, ***kwargs: Any*)

Base class for Device Mixins

DeviceMixinBase(*args, **kwargs)	Base class for apstools Device mixin classes

class apstools.devices.mixin_base.**DeviceMixinBase**(**args: Any*, ***kwargs: Any*)
Base class for apstools Device mixin classes

Mixin classes for Motor Devices

EpicsMotorDialMixin(*args, **kwargs)	add motor record's dial coordinate fields to Device
EpicsMotorEnableMixin(*args, **kwargs)	mixin providing access to motor enable/disable
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
EpicsMotorResolutionMixin(*args, **kwargs)	Add motor record's resolution fields to motor.
EpicsMotorServoMixin(*args, **kwargs)	add motor record's servo loop controls to Device

class apstools.devices.motor_mixins.**EpicsMotorDialMixin**(**args: Any*, ***kwargs: Any*)

add motor record's dial coordinate fields to Device

EXAMPLE:

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

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

class apstools.devices.motor_mixins.**EpicsMotorEnableMixin**(**args: Any*, ***kwargs: Any*)

mixin providing access to motor enable/disable

EXAMPLE:

```
from ophyd import EpicsMotor
from apstools.devices import EpicsMotorEnableMixin

class MyEpicsMotor(EpicsMotorEnableMixin, EpicsMotor): ...

m1 = MyEpicsMotor('xxx:m1', name='m1')
print(m1.enabled)
```

In a bluesky plan:

```
yield from bps.mv(m1.enable_disable, m1.MOTOR_DISABLE)
# ... other activities
yield from bps.mv(m1.enable_disable, m1.MOTOR_ENABLE)
```

disable_motor()

BLOCKING call to disable motor axis

enable_motor()

BLOCKING call to enable motor axis

class apstools.devices.motor_mixins.**EpicsMotorLimitsMixin**(**args: Any*, ***kwargs: Any*)

add motor record HLM & LLM fields & compatibility get_lim() and set_lim()

EXAMPLE:

```
from ophyd import EpicsMotor
from apstools.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 apstools.devices.motor_mixins.**EpicsMotorRawMixin**(**args: Any*, ***kwargs: Any*)

add motor record's raw coordinate fields to Device

EXAMPLE:

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

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

class apstools.devices.motor_mixins.**EpicsMotorResolutionMixin**(**args: Any*, ***kwargs: Any*)

Add motor record's resolution fields to motor.

Usually, a facility will not provide such high-level access to calibration parameters since these are associated with fixed parameters of hardware. For simulators, it is convenient to provide access so that default settings (typically low-resolution) from the IOC can be changed as part of the device setup in bluesky.

EXAMPLE:

```
from ophyd import EpicsMotor
from apstools.devices import EpicsMotorResolutionMixin

class myEpicsMotor(EpicsMotorResolutionMixin, EpicsMotor): pass
m1 = myEpicsMotor('xxx:m1', name='m1')
print(f"resolution={m1.resolution.read()}")
print(f"steps_per_rev={m1.steps_per_rev.read()}")
print(f"units_per_rev={m1.units_per_rev.read()}")
```

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

EXAMPLE:

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

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

PVPositioner that computes done as a soft signal.

PVPositionerSoftDone(*args, **kwargs)	PVPositioner that computes done as a soft signal.
PVPositionerSoftDoneWithStop(*args, **kwargs)	PVPositionerSoftDone with stop() and inposition.

class apstools.devices.positioner_soft_done.**PVPositionerSoftDone**(**args: Any*, ***kwargs: Any*)
PVPositioner that computes done as a soft signal.

PARAMETERS

prefix [str, optional] The device prefix used for all sub-positioners. This is optional as it may be desirable to specify full PV names for PVPositioners.

readback_pv [str, optional] PV prefix of the readback signal. Disregarded if readback attribute is created.

setpoint_pv [str, optional] PV prefix of the setpoint signal. Disregarded if setpoint attribute is created.

tolerance [float, optional] Motion tolerance. The motion is considered *done* when:

```
abs(readback-setpoint) <= tolerance
```

Defaults to `10^(-1*precision)`, where `precision = setpoint.precision`.

update_target [bool] `True` when this object update the `target` Component directly. Use `False` if the `target` Component will be updated externally, such as by the controller when `target` is an `EpicsSignal`. Defaults to `True`.

kwargs : Passed to *ophyd.PVPositioner*

ATTRIBUTES

setpoint [Signal] The setpoint (request) signal

readback [Signal or None] The readback PV (e.g., encoder position PV)

actuate [Signal or None] The actuation PV to set when movement is requested

actuate_value [any, optional] The actuation value, sent to the actuate signal when motion is requested

stop_signal [Signal or None] The stop PV to set when motion should be stopped

stop_value [any, optional] The value sent to stop_signal when a stop is requested

target [Signal] The target value of a move request.

Override (in subclass) with *EpicsSignal* to connect with a PV.

In some controllers (such as temperature controllers), the setpoint may be changed incrementally towards this target value (such as a ramp or controlled trajectory). In such cases, the `target` will be final value while `setpoint` will be the current desired position.

Otherwise, both `setpoint` and `target` will be set to the same value.

(new in apstools 1.5.3)

cb_readback(**args*, ***kwargs*)
Called when readback changes (EPICS CA monitor event).

Computes if the positioner is done moving:

```
done = |readback - setpoint| <= tolerance
```

cb_setpoint(**args*, ***kwargs*)
Called when setpoint changes (EPICS CA monitor event).

When the setpoint is changed, force done=False. For any move, done **must** transition to `!= done_value`, then back to `done_value`.

Without this response, a small move (within tolerance) will not return. Next update of readback will compute `self.done`.

class apstools.devices.positioner_soft_done.**PVPositionerSoftDoneWithStop**(**args: Any*, ***kwargs: Any*)
PVPositionerSoftDone with stop() and inposition.

The *stop()* method sets the setpoint to the immediate readback value (only when `inposition` is `True`). This stops the positioner at the current position.

property inposition
Report (boolean) if positioner is done.

stop(*, *success=False*)
Hold the current readback when stop() is called and not *inposition()*.

Generalized ophyd Device base class for preamplifiers.

PreamplifierBaseDevice(*args, **kwargs)	Generalized interface (base class) for preamplifiers.

class apstools.devices.preamp_base.**PreamplifierBaseDevice**(**args: Any*, ***kwargs: Any*)
Generalized interface (base class) for preamplifiers.

All subclasses of `PreamplifierBaseDevice` must define how to update the gain with the correct value from the amplifier. An example is `SRS570_PreAmplifier`.

See https://github.com/BCDA-APS/apstools/issues/544

PTC10 Programmable Temperature Controller

The PTC10 is a programmable temperature controller from SRS (Stanford Research Systems). The PTC10 is a modular system consisting of a base unit and provision for addition of add-on boards.

A single, complete `ophyd.Device` subclass will not describe all variations that could be installed. But the add-on boards each allow standardization. Each installation must build a custom class that matches their hardware configuration. The APS USAXS instrument has created a custom class based on the *ophyd.PVPositioner* to use their *PTC10* as a temperature *positioner*.

PTC10AioChannel(*args, **kwargs)	SRS PTC10 AIO module
PTC10RtdChannel(*args, **kwargs)	SRS PTC10 RTD module channel
PTC10TcChannel(*args, **kwargs)	SRS PTC10 Tc (thermocouple) module channel

continues on next page

Table 30 – continued from previous page

PTC10PositionerMixin(*args, **kwargs)	Mixin so SRS PTC10 can be used as a (temperature) positioner.

see https://www.thinksrs.com/products/ptc10.html

EXAMPLE:

```
from ophyd import PVPositioner

class MyPTC10(PTC10PositionerMixin, PVPositioner):
    readback = Component(EpicsSignalRO, "2A:temperature", kind="hinted")
    setpoint = Component(EpicsSignalWithRBV, "5A:setPoint", kind="hinted")

    rtd = Component(PTC10RtdChannel, "3A:")
    pid = Component(PTC10AioChannel, "5A:")

ptc10 = MyPTC10("IOC_PREFIX:ptc10:", name="ptc10")
ptc10.report_dmov_changes.put(True)  # a diagnostic
ptc10.tolerance.put(1.0)  # done when |readback-setpoint|<=tolerance
```

New in apstools 1.5.3.

class apstools.devices.ptc10_controller.**PTC10AioChannel**(**args: Any*, ***kwargs: Any*)
SRS PTC10 AIO module

class apstools.devices.ptc10_controller.**PTC10PositionerMixin**(**args: Any*, ***kwargs: Any*)
Mixin so SRS PTC10 can be used as a (temperature) positioner.

cb_readback(*args, **kwargs)	Called when readback changes (EPICS CA monitor event).
cb_setpoint(*args, **kwargs)	Called when setpoint changes (EPICS CA monitor event).
inposition	Report (boolean) if positioner is done.
stop(*[, success])	Hold the current readback when the stop() method is called and not done.

cb_readback(**args*, ***kwargs*)
Called when readback changes (EPICS CA monitor event).

cb_setpoint(**args*, ***kwargs*)
Called when setpoint changes (EPICS CA monitor event).

When the setpoint is changed, force `done=False`. For any move, `done` MUST change to `!= done_value`, then change back to `done_value (True)`. Without this response, a small move (within tolerance) will not return. Next update of readback will compute `self.done`.

property inposition
Report (boolean) if positioner is done.

stop(*, *success=False*)
Hold the current readback when the stop() method is called and not done.

class apstools.devices.ptc10_controller.**PTC10RtdChannel**(**args: Any*, ***kwargs: Any*)
SRS PTC10 RTD module channel

class apstools.devices.ptc10_controller.**PTC10TcChannel**(**args: Any*, ***kwargs: Any*)
SRS PTC10 Tc (thermocouple) module channel

Scaler support

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

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

Note: For *ScalerCH*, use *scaler.select_channels(None)* instead of this code. (Applies only to *ophyd.scaler.ScalerCH* in releases after 2019-02-27.)

Shutters

ApsPssShutter(*args, **kwargs)	APS PSS shutter
ApsPssShutterWithStatus(*args, **kwargs)	APS PSS shutter with separate status PV
EpicsMotorShutter(*args, **kwargs)	Shutter, implemented with an EPICS motor moved between two positions
EpicsOnOffShutter(*args, **kwargs)	Shutter using a single EPICS PV moved between two positions
OneSignalShutter(*args, **kwargs)	Shutter Device using one Signal for open and close.
ShutterBase(*args, **kwargs)	Base class for all shutter Devices.
SimulatedApsPssShutterWithStatus(*args, **kwargs)	Simulated APS PSS shutter

class apstools.devices.shutters.**ApsPssShutter**(**args: Any*, ***kwargs: Any*)
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)

Since there is no direct indication that a shutter has moved, the `state` property will always return *unknown* and the `isOpen` and `isClosed` properties will always return *False*.

A consequence of the unknown state is that the shutter will always be commanded to move (and wait the `delay_s` time), even if it is already at that position. This device could keep track of the last commanded position, but that is not guaranteed to be true since the shutter could be moved from other software.

The default `delay_s` has been set at *1.2 s* to allow for shutter motion. Change this as desired. Advise if this default should be changed.

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.addOpenValue(“o”) shutter_a.addCloseValue(“x”) shutter_a.set(“o”) shutter_a.set(“x”)

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

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

property state
is shutter “open”, “close”, or “unknown”?

class apstools.devices.shutters.**ApsPssShutterWithStatus**(**args: Any*, ***kwargs: Any*)
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")

A_shutter.open()
A_shutter.close()

or

A_shutter.set("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))

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

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

property state
is shutter “open”, “close”, or “unknown”?

wait_for_state(*target*, *timeout=10*, *poll_s=0.01*)
wait for the PSS state to reach a desired target

PARAMETERS

target *[str]* : list of strings containing acceptable values

timeout *non-negative number* : maximum amount of time (seconds) to wait for PSS state to reach target

poll_s *non-negative number* : Time to wait (seconds) in first polling cycle. After first poll, this will be increased by _poll_factor_ up to a maximum time of _poll_s_max_.

class apstools.devices.shutters.**EpicsMotorShutter**(**args: Any*, ***kwargs: Any*)
Shutter, implemented with an EPICS motor moved between two positions

EXAMPLE:

```
tomo_shutter = EpicsMotorShutter("2bma:m23", name="tomo_shutter")
tomo_shutter.close_value = 1.0      # default
tomo_shutter.open_value = 0.0       # default
tomo_shutter.tolerance = 0.01       # 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

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

property state
is shutter “open”, “close”, or “unknown”?

class apstools.devices.shutters.**EpicsOnOffShutter**(**args: Any*, ***kwargs: Any*)
Shutter using a single 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.close_value = 0      # default
bit_shutter.open_value = 1       # default
bit_shutter.open()
bit_shutter.close()

# or, when used in a plan
def planA():
    yield from mv(bit_shutter, "open")
    yield from mv(bit_shutter, "close")
```

class apstools.devices.shutters.**OneSignalShutter**(**args: Any*, ***kwargs: Any*)

Shutter Device using one Signal for open and close.

PARAMETERS

signal `EpicsSignal` or `Signal` : (override in subclass) The `signal` is the comunication to the hardware. In a subclass, the hardware may have more than one communication channel to use. See the `ApsPssShutter` as an example.

See `ShutterBase` for more parameters.

EXAMPLE

Create a simulated shutter:

shutter = OneSignalShutter(name="shutter")

open the shutter (interactively):

shutter.open()

Check the shutter is open:

In [144]: shutter.isOpen Out[144]: True

Use the shutter in a Bluesky plan. Set a post-move delay time of 1.0 seconds. Be sure to use `yield from`, such as:

```
def in_a_plan(shutter):
    shutter.delay_s = 1.0
    t0 = time.time()
    print("Shutter state: " + shutter.state, time.time()-t0)
    yield from bps.abs_set(shutter, "open", wait=True)    # wait for completion is␣
↪optional
    print("Shutter state: " + shutter.state, time.time()-t0)
    yield from bps.mv(shutter, "open")    # do it again
    print("Shutter state: " + shutter.state, time.time()-t0)
    yield from bps.mv(shutter, "close")    # ALWAYS waits for completion
    print("Shutter state: " + shutter.state, time.time()-t0)

RE(in_a_plan(shutter))
```

which gives this output:

Shutter state: close 1.7642974853515625e-05 Shutter state: open 1.0032124519348145 Shutter state: open 1.0057861804962158 Shutter state: close 2.009695529937744

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.addOpenValue(“o”) shutter.addCloseValue(“x”) shutter.set(“o”) shutter.set(“x”)

close()
BLOCKING: request shutter to close, called by set()

open()
BLOCKING: request shutter to open, called by set()

property state
is shutter “open”, “close”, or “unknown”?

class apstools.devices.shutters.**ShutterBase**(**args: Any*, ***kwargs: Any*)
Base class for all shutter Devices.

PARAMETERS

value *str* : any from `self.choices` (typically “open” or “close”)

valid_open_values *[str]* : A list of lower-case text values that are acceptable for use with the `set()` command to open the shutter.

valid_close_values *[str]* : A list of lower-case text values that are acceptable for use with the `set()` command to close the shutter.

open_value *number* : The actual value to send to open `signal` to open the shutter. (default = 1)

close_value *number* : The actual value to send to close `signal` to close the shutter. (default = 0)

delay_s *float* : time to wait (s) after move is complete, does not wait if shutter already in position (default = 0)

busy *Signal* : (internal) tells if a move is in progress

unknown_state *str* : (constant) Text reported by `state` when not open or closed. cannot move to this position (default = “unknown”)

addCloseValue(*text*)
a synonym to close the shutter, use with set()

addOpenValue(*text*)
a synonym to open the shutter, use with set()

property choices
return list of acceptable choices for set()

close()
BLOCKING: request shutter to close, called by `set()`.

Must implement in subclass of ShutterBase()

EXAMPLE:

```
if not self.isClosed:
    self.signal.put(self.close_value)
    if self.delay_s > 0:
        time.sleep(self.delay_s)    # blocking call OK here
```

inPosition(*target*)
is the shutter at the target position?

property isClosed
is the shutter closed?

property isOpen
is the shutter open?

lowerCaseString(*value*)
ensure any given value is a lower-case string

open()
BLOCKING: request shutter to open, called by `set()`.

Must implement in subclass of ShutterBase()

EXAMPLE:

```
if not self.isOpen:
    self.signal.put(self.open_value)
    if self.delay_s > 0:
        time.sleep(self.delay_s)    # blocking call OK here
```

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

PARAMETERS

value *str* : any from `self.choices` (typically "open" or "close")

kwargs *dict* : ignored at this time

property state
returns `open`, `close`, or `unknown`

Must implement in subclass of ShutterBase()

EXAMPLE:

```
if self.signal.get() == self.open_value:
    result = self.valid_open_values[0]
elif self.signal.get() == self.close_value:
    result = self.valid_close_values[0]
else:
    result = self.unknown_state
return result
```

validTarget(*target*, *should_raise=True*)
return whether (or not) target value is acceptable for self.set()

raise ValueError if not acceptable (default)

class apstools.devices.shutters.**SimulatedApsPssShutterWithStatus**(**args: Any*, ***kwargs: Any*)
Simulated APS PSS shutter

EXAMPLE:

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

property state
is shutter "open", "close", or "unknown"?

wait_for_state(*target*, *timeout=10*, *poll_s=0.01*)
wait for the PSS state to reach a desired target

PARAMETERS

target *[str]* : list of strings containing acceptable values

timeout *non-negative number* : Ignored in the simulation.

poll_s *non-negative number* : Ignored in the simulation.

Ophyd support for Stanford Research Systems 570 preamplifier from synApps

Public Structures

SRS570_PreAmplifier(*args, **kwargs)	Ophyd support for Stanford Research Systems 570 preamplifier from synApps.

This device connects with the SRS570 support from synApps. (https://github.com/epics-modules/ip/blob/master/ipApp/Db/SR570.db)

The SRS570 synApps support is part of the `ip` module: https://htmlpreview.github.io/?https://raw.githubusercontent.com/epics-modules/ip/R3-6-1/documentation/swaitRecord.html

see https://github.com/epics-modules/ip

class apstools.devices.srs570_preamplifier.**SRS570_PreAmplifier**(**args: Any*, ***kwargs: Any*)
Ophyd support for Stanford Research Systems 570 preamplifier from synApps.

cb_gain(**args*, ***kwargs*)
Called when sensitivity changes (EPICS CA monitor event).

property computed_gain
Amplifier gain (A/V), as floating-point number.

Struck 3820

Struck3820(*args, **kwargs)	Struck/SIS 3820 Multi-Channel Scaler (as used by USAXS)

class apstools.devices.struck3820.**Struck3820**(**args: Any*, ***kwargs: Any*)
Struck/SIS 3820 Multi-Channel Scaler (as used by USAXS)

Synthetic pseudo-Voigt function

EXAMPLES:

Listing 6: Simple example of SynPseudoVoigt().

```
from apstools.devices import SynPseudoVoigt
from ophyd.sim import motor
det = SynPseudoVoigt('det', motor, 'motor',
    center=0, eta=0.5, scale=1, sigma=1, bkg=0)

# scan the "det" peak with the "motor" positioner
# RE(bp.scan([det], motor, -2, 2, 41))
```

Listing 7: Example of SynPseudoVoigt() with randomized values.

```
import numpy as np
from apstools.devices 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())

# scan the "synthetic_pseudovoigt" peak with the "m1" positioner
# RE(bp.scan([synthetic_pseudovoigt], m1, -2, 0, 219))
```

SynPseudoVoigt(*args, **kwargs)	Evaluate a point on a pseudo-Voigt based on the value of a motor.

class apstools.devices.synth_pseudo_voigt.**SynPseudoVoigt**(**args: Any*, ***kwargs: Any*)

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 positioner : The independent coordinate

motor_field *str* : name of *motor*

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" or "uniform" or None : Add noise to the result.

noise_multiplier *float* : Only relevant for 'uniform' noise. Multiply the random amount of noise by 'noise_multiplier'

Tracking Signal for Device coordination

TrackingSignal(*args, **kwargs)	Non-EPICS signal for use when coordinating Device actions.

class apstools.devices.tracking_signal.**TrackingSignal**(**args: Any*, ***kwargs: Any*)

Non-EPICS signal for use when coordinating Device actions.

Signal to decide if undulator will be tracked while changing the monochromator energy.

check_value(*value*)
Check if the value is a boolean.

RAISES

ValueError

XIA PF4 Filters

Pf4FilterSingle(*args, **kwargs)	XIA PF4 Filter: one set of 4 filters (A).
Pf4FilterDual(*args, **kwargs)	XIA PF4 Filter: two sets of 4 filters (A, B).
Pf4FilterTriple(*args, **kwargs)	XIA PF4 Filter: three sets of 4 filters (A, B, C).
Pf4FilterCommon(*args, **kwargs)	XIA PF4 filters - common support.
Pf4FilterBank(*args, **kwargs)	A single module of XIA PF4 filters (4-blades).
DualPf4FilterBox(*args, **kwargs)	LEGACY (use Pf4FilterDual now): Dual (Al, Ti) Xia PF4 filter boxes

class apstools.devices.xia_pf4.**DualPf4FilterBox**(**args: Any*, ***kwargs: Any*)
LEGACY (use Pf4FilterDual now): Dual (Al, Ti) Xia PF4 filter boxes

Support from synApps (using Al, Ti foils)

EXAMPLE:

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

class apstools.devices.xia_pf4.**Pf4FilterBank**(**args: Any*, ***kwargs: Any*)
A single module of XIA PF4 filters (4-blades).

EXAMPLES:

```
pf4B = Pf4FilterBank("ioc:pf4:", name="pf4B", bank="B")

# -or-

class MyTriplePf4(Pf4FilterCommon):
    A = Component(Pf4FilterBank, "", bank="A")
    B = Component(Pf4FilterBank, "", bank="B")
    C = Component(Pf4FilterBank, "", bank="C")

pf4 = MyTriplePf4("ioc:pf4:", name="pf4")
```

See https://github.com/epics-modules/optics/blob/master/opticsApp/Db/pf4bank.db

class apstools.devices.xia_pf4.**Pf4FilterCommon**(**args: Any*, ***kwargs: Any*)
XIA PF4 filters - common support.

Use `Pf4FilterCommon` to build support for a configuration of PF4 filters (such as 3 or 4 filter banks).

EXAMPLE:

```
class MyTriplePf4(Pf4FilterCommon):
    A = Component(Pf4FilterBank, "", bank="A")
    B = Component(Pf4FilterBank, "", bank="B")
```

(continues on next page)

(continued from previous page)

```
    C = Component(Pf4FilterBank, "", bank="C")

pf4 = MyTriplePf4("ioc:pf4:", name="pf4")
```

See https://github.com/epics-modules/optics/blob/master/opticsApp/Db/pf4common.db

class apstools.devices.xia_pf4.**Pf4FilterDual**(**args: Any*, ***kwargs: Any*)
XIA PF4 Filter: two sets of 4 filters (A, B).

B
alias of *apstools.devices.xia_pf4.Pf4FilterBank*

class apstools.devices.xia_pf4.**Pf4FilterSingle**(**args: Any*, ***kwargs: Any*)
XIA PF4 Filter: one set of 4 filters (A).

A
alias of *apstools.devices.xia_pf4.Pf4FilterBank*

class apstools.devices.xia_pf4.**Pf4FilterTriple**(**args: Any*, ***kwargs: Any*)
XIA PF4 Filter: three sets of 4 filters (A, B, C).

C
alias of *apstools.devices.xia_pf4.Pf4FilterBank*

XIA Slit from EPICS synApps optics: xia_slit.db

Coordinates (viewing from detector towards source):

```
    top
inb     out
    bot
```

Each blade[1] (in the XIA slit controller) travels in a _cylindrical_ coordinate system. Positive motion moves a blade **outwards** from the center with a backlash correction. No backlash correction is applied for negative motion (as the blades close). Size and center are computed by the underlying EPICS support.

hsize = inb + out vsize = top + bot

USAGE:

slit = XiaSlitController("IOC:hsc1:", name="slit") print(slit.geometry)

XiaSlit2D(*args, **kwargs)	EPICS synApps optics xia_slit.db 2D support: inb out bot top ...

class apstools.devices.xia_slit.**XiaSlit2D**(**args: Any*, ***kwargs: Any*)
EPICS synApps optics xia_slit.db 2D support: inb out bot top ...

bot
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

property geometry
Return the slit 2D size and center as a namedtuple.

[1] Note that the blade names here are different than the EPICS support. The difference is to make the names of the blades consistent with other slits with the Bluesky framework.

hcenter
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

hsize
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

inb
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

out
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

top
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

vcenter
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

vsize
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

2.4.3 File Writers

The file writer callbacks are:

FileWriterCallbackBase(*args, **kwargs)	Base class for filewriter callbacks.
NXWriterAPS(*args, **kwargs)	Customize `NXWriter` with APS-specific content.
NXWriter(*args, **kwargs)	General class for writing HDF5/NeXus file (using only NeXus base classes).
SpecWriterCallback([filename, auto_write, ...])	Collect data from Bluesky RunEngine documents to write as SPEC data.

Overview

Each filewriter can be used as a callback to the Bluesky RunEngine for live data acquisition or later as a handler for a document set from the databroker. Methods are provided to handle each document type. The callback method, `receiver(document_type, document)`, receives the set of documents, one-by-one, and sends them to the appropriate handler. Once the `stop` document is received, the `writer()` method is called to write the output file.

Examples

Write SPEC file automatically from data acquisition:

```
specwriter = apstools.callbacks.SpecWriterCallback()
RE.subscribe(specwriter.receiver)
```

Write NeXus file automatically from data acquisition:

```
nxwriter = apstools.callbacks.NXWriter()
RE.subscribe(nxwriter.receiver)
```

Write APS-specific NeXus file automatically from data acquisition:

```
nxwriteraps = apstools.callbacks.NXWriterAPS()
RE.subscribe(nxwriteraps.receiver)
```

Programmer's Note

Subclassing from `object` (or no superclass) avoids the need to import `bluesky.callbacks.core.CallbackBase`. One less import when only accessing the Databroker. The *only* advantage to subclassing from `CallbackBase` seems to be a simpler setup call to `RE.subscribe()`.

superclass	subscription code
object	`RE.subscribe(specwriter.receiver)`
CallbackBase	`RE.subscribe(specwriter)`

HDF5/NeXus File Writers

FileWriterCallbackBase

Base class for filewriter callbacks.

Applications should subclass and rewrite the `writer()` method.

The local buffers are cleared when a start document is received. Content is collected here from each document until the stop document. The content is written once the stop document is received.

Output File Name and Path

The output file will be written to the file named in `self.file_name`. (Include the output directory if different from the current working directory.)

When `self.file_name` is `None` (the default), the *make_file_name()* method will construct the file name using a combination of the date and time (from the `start` document), the `start` document `uid`, and the `scan_id`. The default file extension (given in `NEXUS_FILE_EXTENSION`) is used in *make_file_name()*. The directory will be `self.file_path` (or the current working directory if `self.file_path` is `None` which is the default).

Either specify `self.file_name` or override *make_file_name()* in a subclass to change the procedure for default output file names.

Metadata

Almost all metadata keys (additional attributes added to the run's `start` document) are completely optional. Certain keys are specified by the RunEngine, some keys are specified by the plan (or plan support methods), and other keys are supplied by the user or the instrument team.

These are the keys used by this callback to help guide how information is stored in a NeXus HDF5 file structure.

key	creator	how is it used
detectors	inconsistent	name(s) of the signals used as detectors
motors	inconsistent	synonym for `positioners`
plan_args	inconsistent	parameters (arguments) given
plan_name	inconsistent	name of the plan used to collect data
positioners	inconsistent	name(s) of the signals used as positioners
scan_id	RunEngine	incrementing number of the run, user can reset
uid	RunEngine	unique identifier of the run
versions	instrument	documents the software versions used to collect data

For more information about Bluesky *events* and document types, see https://blueskyproject.io/event-model/data-model.html.

NXWriter

General class for writing HDF5/NeXus file (using only NeXus base classes).

One scan is written to one HDF5/NeXus file.

Output File Name and Path

The output file will be written to the file named in `self.file_name`. (Include the output directory if different from the current working directory.)

When `self.file_name` is `None` (the default), the *make_file_name()* method will construct the file name using a combination of the date and time (from the *start* document), the `start` document `uid`, and the `scan_id`. The default file extension (given in `NEXUS_FILE_EXTENSION`) is used in *make_file_name()*. The directory will be `self.file_path` (or the current working directory if `self.file_path` is `None` which is the default).

Either specify `self.file_name` of override *make_file_name()* in a subclass to change the procedure for default output file names.

Metadata

Almost all metadata keys (additional attributes added to the run's `start` document) are completely optional. Certain keys are specified by the RunEngine, some keys are specified by the plan (or plan support methods), and other keys are supplied by the user or the instrument team.

These are the keys used by this callback to help guide how information is stored in a NeXus HDF5 file structure.

key	creator	how is it used
detectors	inconsistent	name(s) of the signals used as plottable values
motors	inconsistent	synonym for `positioners`
plan_args	inconsistent	parameters (arguments) given
plan_name	inconsistent	name of the plan used to collect data
positioners	inconsistent	name(s) of the positioners used for plotting
scan_id	RunEngine	incrementing number of the run, user can reset
subtitle	user	-tba-
title	user	/entry/title
uid	RunEngine	unique identifier of the run
versions	instrument	documents the software versions used to collect data

Notes:

1. `detectors[0]` will be used as the `/entry/data@signal` attribute
2. the *complete* list in `positioners` will be used as the `/entry/data@axes` attribute

NXWriterAPS

Customize *NXWriter* with APS-specific content.

- Adds `/entry/instrument/undulator` group if metadata exists.
- Adds APS information to `/entry/instrument/source` group.

APS instruments should subclass *NXWriterAPS* to make customizations for specific plans or other considerations.

HDF5/NeXus File Structures

Bluesky stores a wealth of information about a measurement in a *run*. Raw data from the Bluesky *run* is stored in the HDF5/NeXus structure under the `/entry/instrument/bluesky` group as shown in this example:

```
bluesky:NXnote
  @NX_class = NXnote
  @target = /entry/instrument/bluesky
  plan_name --> /entry/instrument/bluesky/metadata/plan_name
  uid --> /entry/instrument/bluesky/metadata/run_start_uid
```

The NeXus structure is built using links to the raw data in `/entry/instrument/bluesky`.

Metadata

Metadata from the `start` document is stored in the `metadata` subgroup as shown in this example:

```
metadata:NXnote
  @NX_class = NXnote
  @target = /entry/instrument/bluesky/metadata
  beamline_id:NX_CHAR = b'APS USAXS 9-ID-C'
    @target = /entry/instrument/bluesky/metadata/beamline_id
  datetime:NX_CHAR = b'2019-05-02 17:45:33.904824'
    @target = /entry/instrument/bluesky/metadata/datetime
  detectors:NX_CHAR = b'- I0_USAXS\n'
    @target = /entry/instrument/bluesky/metadata/detectors
    @text_format = yaml
  hints:NX_CHAR = b'dimensions:\n- - - m_stage_r\n  - primary\n'
    @target = /entry/instrument/bluesky/metadata/hints
    @text_format = yaml
  login_id:NX_CHAR = b'usaxs@usaxscontrol.xray.aps.anl.gov'
    @target = /entry/instrument/bluesky/metadata/login_id
  motors:NX_CHAR = b'- m_stage_r\n'
    @target = /entry/instrument/bluesky/metadata/motors
    @text_format = yaml
  pid:NX_INT64[] =
    @target = /entry/instrument/bluesky/metadata/pid
```

(continues on next page)

(continued from previous page)

```
    plan_name:NX_CHAR = b'tune_mr'
      @target = /entry/instrument/bluesky/metadata/plan_name
    plan_type:NX_CHAR = b'generator'
      @target = /entry/instrument/bluesky/metadata/plan_type
    proposal_id:NX_CHAR = b'testing Bluesky installation'
      @target = /entry/instrument/bluesky/metadata/proposal_id
    purpose:NX_CHAR = b'tuner'
      @target = /entry/instrument/bluesky/metadata/purpose
    run_start_uid:NX_CHAR = 2ffe4d87-9f0c-464a-9d14-213ec71afaf7
      @long_name = bluesky run uid
      @target = /entry/instrument/bluesky/metadata/run_start_uid
    tune_md:NX_CHAR = b"initial_position: 8.824977\ntime_iso8601: '2019-05-02 17:45:33.
↪923544'\nwidth: -0.004\n"
      @target = /entry/instrument/bluesky/metadata/tune_md
      @text_format = yaml
    tune_parameters:NX_CHAR = b'initial_position: 8.824977\nnum: 31\npeak_choice: com\
↪nwidth: -0.004\nx_axis: m_stage_r\ny_axis: I0_USAXS\n'
      @target = /entry/instrument/bluesky/metadata/tune_parameters
      @text_format = yaml
    uid --> /entry/instrument/bluesky/run_start_uid
```

Note that complex structures (lists and dictionaries) are written as YAML. YAML is easily converted back into the python structure using `yaml.load(yaml_text)` where `yaml_text` is the YAML text from the HDF5 file.

Streams

Data from each ophyd signal in a *document stream* is stored as datasets within a `NXdata`[1] subgroup of that stream. Bluesky collects value(s) and timestamp(s) which are stored in the datasets `value` and `EPOCH`, respectively. Since `EPOCH` is the absolute number of seconds from some time in the deep past, an additional `time` dataset repeats this data as time *relative* to the first time. (This makes it much easier for some visualization programs to display value v. time plots.) Additional information, as available (such as units), is added as NeXus attributes. The `@target` attribute is automatically added to simplify linking any item into the NeXus tree structure.

For example, the main data in a run is usually stored in the `primary` stream. Here, we show the tree structure for one signal (`I0_USAXS`) from the primary stream:

```
  primary:NXnote
    @NX_class = NXnote
    @target = /entry/instrument/bluesky/streams/primary
    @uid = 90489e9b-d66e-4753-8c4f-849e7a809aeb
    I0_USAXS:NXdata
      @NX_class = NXdata
      @axes = time
      @signal = value
      @signal_type = detector
      @target = /entry/instrument/bluesky/streams/primary/I0_USAXS
      EPOCH:NX_FLOAT64[31] = [1556837134.972796, 1556837135.589462, 1556837135.989462, '.
↪..', 1556837147.656129]
        @long_name = epoch time (s)
        @target = /entry/instrument/bluesky/streams/primary/I0_USAXS/EPOCH
```

(continues on next page)

[1] `NXdata` is used since some visualization tools recognize it to make a plot.

(continued from previous page)

```
        @units = s
      time:NX_FLOAT64[31] = [0.0, 0.6166660785675049, 1.0166659355163574, '...', 12.
↪683332920074463]
        @long_name = time since first data (s)
        @start_time = 1556837134.972796
        @start_time_iso = 2019-05-02T17:45:34.972796
        @target = /entry/instrument/bluesky/streams/primary/I0_USAXS/time
        @units = s
      value:NX_FLOAT64[31] = [127.0, 126.0, 127.0, '...', 127.0]
        @long_name = I0_USAXS
        @lower_ctrl_limit = 0.0
        @precision = 0
        @signal_type = detector
        @source = PV:9idcLAX:vsc:c0.S2
        @target = /entry/instrument/bluesky/streams/primary/I0_USAXS/value
        @units =
        @upper_ctrl_limit = 0.0
```

Baseline

In Bluesky, *baseline* streams record the value (and timestamp) of a signal at the start and end of the run. Similar to the handling for streams (above), a subgroup is created for each baseline stream. The datasets include `value`, `EPOCH`, `time` (as above) and `value_start` and `value_end`. Here's an example:

```
  baseline:NXnote
    @NX_class = NXnote
    @target = /entry/instrument/bluesky/streams/baseline
    @uid = f5fce6ac-f3fa-4c34-b11d-9e33c263cd20
    aps_current:NXdata
      @NX_class = NXdata
      @axes = time
      @signal = value
      @target = /entry/instrument/bluesky/streams/baseline/aps_current
      EPOCH:NX_FLOAT64[2] = [1556837133.753769, 1556837147.753723]
        @long_name = epoch time (s)
        @target = /entry/instrument/bluesky/streams/baseline/aps_current/EPOCH
        @units = s
      time:NX_FLOAT64[2] = [0.0, 13.999953985214233]
        @long_name = time since first data (s)
        @start_time = 1556837133.753769
        @start_time_iso = 2019-05-02T17:45:33.753769
        @target = /entry/instrument/bluesky/streams/baseline/aps_current/time
        @units = s
      value:NX_FLOAT64[2] = [0.004512578244000032, 0.003944485484000011]
        @long_name = aps_current
        @lower_ctrl_limit = 0.0
        @precision = 1
        @source = PV:S:SRcurrentAI
        @target = /entry/instrument/bluesky/streams/baseline/aps_current/value
        @units = mA
        @upper_ctrl_limit = 310.0
```

(continues on next page)

(continued from previous page)

```
      value_end:NX_FLOAT64[] =
        @long_name = aps_current
        @lower_ctrl_limit = 0.0
        @precision = 1
        @source = PV:S:SRcurrentAI
        @target = /entry/instrument/bluesky/streams/baseline/aps_current/value_end
        @units = mA
        @upper_ctrl_limit = 310.0
      value_start:NX_FLOAT64[] =
        @long_name = aps_current
        @lower_ctrl_limit = 0.0
        @precision = 1
        @source = PV:S:SRcurrentAI
        @target = /entry/instrument/bluesky/streams/baseline/aps_current/value_start
        @units = mA
        @upper_ctrl_limit = 310.0
```

Full Structure

The full structure of the example HDF5/NeXus file (omitting the attributes and array data for brevity) is shown next. You can see that most of the NeXus structure is completed by making links to data from either `/entry/instrument/bluesky/metadata` or `/entry/instrument/bluesky/streams`:

```
20190502-174533-S00108-2ffe4d8.hdf : NeXus data file
  entry:NXentry
    duration:NX_FLOAT64[] = [ ... ]
    end_time:NX_CHAR = 2019-05-02T17:45:48.078618
    entry_identifier --> /entry/instrument/bluesky/uid
    plan_name --> /entry/instrument/bluesky/metadata/plan_name
    program_name:NX_CHAR = bluesky
    start_time:NX_CHAR = 2019-05-02T17:45:33.937294
    title:NX_CHAR = tune_mr-S0108-2ffe4d8
    contact:NXuser
      affiliation --> /entry/instrument/bluesky/streams/baseline/bss_user_info_
↪institution/value_start
      email --> /entry/instrument/bluesky/streams/baseline/bss_user_info_email/value_
↪start
      facility_user_id --> /entry/instrument/bluesky/streams/baseline/bss_user_info_
↪badge/value_start
      name --> /entry/instrument/bluesky/streams/baseline/bss_user_info_contact/value_
↪start
      role:NX_CHAR = contact
    data:NXdata
      EPOCH --> /entry/instrument/bluesky/streams/primary/scaler0_time/time
      I0_USAXS --> /entry/instrument/bluesky/streams/primary/I0_USAXS/value
      m_stage_r --> /entry/instrument/bluesky/streams/primary/m_stage_r/value
      m_stage_r_soft_limit_hi --> /entry/instrument/bluesky/streams/primary/m_stage_r_
↪soft_limit_hi/value
      m_stage_r_soft_limit_lo --> /entry/instrument/bluesky/streams/primary/m_stage_r_
↪soft_limit_lo/value
      m_stage_r_user_setpoint --> /entry/instrument/bluesky/streams/primary/m_stage_r_
↪user_setpoint/value
```

(continues on next page)

(continued from previous page)

```
      scaler0_display_rate --> /entry/instrument/bluesky/streams/primary/scaler0_display_
↪rate/value
      scaler0_time --> /entry/instrument/bluesky/streams/primary/scaler0_time/value
    instrument:NXinstrument
      bluesky:NXnote
        plan_name --> /entry/instrument/bluesky/metadata/plan_name
        uid --> /entry/instrument/bluesky/metadata/run_start_uid
        metadata:NXnote
          APSTOOLS_VERSION:NX_CHAR = b'1.1.0'
          BLUESKY_VERSION:NX_CHAR = b'1.5.2'
          EPICS_CA_MAX_ARRAY_BYTES:NX_CHAR = b'1280000'
          EPICS_HOST_ARCH:NX_CHAR = b'linux-x86_64'
          OPHYD_VERSION:NX_CHAR = b'1.3.3'
          beamline_id:NX_CHAR = b'APS USAXS 9-ID-C'
          datetime:NX_CHAR = b'2019-05-02 17:45:33.904824'
          detectors:NX_CHAR = b'- I0_USAXS\n'
          hints:NX_CHAR = b'dimensions:\n- - - m_stage_r\n  - primary\n'
          login_id:NX_CHAR = b'usaxs@usaxscontrol.xray.aps.anl.gov'
          motors:NX_CHAR = b'- m_stage_r\n'
          pid:NX_INT64[] = [ ... ]
          plan_name:NX_CHAR = b'tune_mr'
          plan_type:NX_CHAR = b'generator'
          proposal_id:NX_CHAR = b'testing Bluesky installation'
          purpose:NX_CHAR = b'tuner'
          run_start_uid:NX_CHAR = 2ffe4d87-9f0c-464a-9d14-213ec71afaf7
          tune_md:NX_CHAR = b"initial_position: 8.824977\ntime_iso8601: '2019-05-02␣
↪17:45:33.923544'\nwidth: -0.004\n"
          tune_parameters:NX_CHAR = b'initial_position: 8.824977\nnum: 31\npeak_choice:␣
↪com\nwidth: -0.004\nx_axis: m_stage_r\ny_axis: I0_USAXS\n'
          uid --> /entry/instrument/bluesky/run_start_uid
        streams:NXnote
          baseline:NXnote
            aps_current:NXdata
              EPOCH:NX_FLOAT64[2] = [ ... ]
              time:NX_FLOAT64[2] = [ ... ]
              value:NX_FLOAT64[2] = [ ... ]
              value_end:NX_FLOAT64[] = [ ... ]
              value_start:NX_FLOAT64[] = [ ... ]
            aps_fill_number:NXdata
              EPOCH:NX_FLOAT64[2] = [ ... ]
              time:NX_FLOAT64[2] = [ ... ]
              value:NX_FLOAT64[2] = [ ... ]
              value_end:NX_FLOAT64[] = [ ... ]
              value_start:NX_FLOAT64[] = [ ... ]
            aps_global_feedback:NXdata
              EPOCH:NX_FLOAT64[2] = [ ... ]
              time:NX_FLOAT64[2] = [ ... ]
              value:NX_CHAR[3,3] = ["Off", "Off"]
              value_end:NX_CHAR = b'Off'
              value_start:NX_CHAR = b'Off'
            # many baseline groups omitted for brevity
          primary:NXnote
```

(continues on next page)

(continued from previous page)

```
            I0_USAXS:NXdata
              EPOCH:NX_FLOAT64[31] = [ ... ]
              time:NX_FLOAT64[31] = [ ... ]
              value:NX_FLOAT64[31] = [ ... ]
            m_stage_r:NXdata
              EPOCH:NX_FLOAT64[31] = [ ... ]
              time:NX_FLOAT64[31] = [ ... ]
              value:NX_FLOAT64[31] = [ ... ]
            m_stage_r_soft_limit_hi:NXdata
              EPOCH:NX_FLOAT64[31] = [ ... ]
              time:NX_FLOAT64[31] = [ ... ]
              value:NX_FLOAT64[31] = [ ... ]
            m_stage_r_soft_limit_lo:NXdata
              EPOCH:NX_FLOAT64[31] = [ ... ]
              time:NX_FLOAT64[31] = [ ... ]
              value:NX_FLOAT64[31] = [ ... ]
            m_stage_r_user_setpoint:NXdata
              EPOCH:NX_FLOAT64[31] = [ ... ]
              time:NX_FLOAT64[31] = [ ... ]
              value:NX_FLOAT64[31] = [ ... ]
            scaler0_display_rate:NXdata
              EPOCH:NX_FLOAT64[31] = [ ... ]
              time:NX_FLOAT64[31] = [ ... ]
              value:NX_FLOAT64[31] = [ ... ]
            scaler0_time:NXdata
              EPOCH:NX_FLOAT64[31] = [ ... ]
              time:NX_FLOAT64[31] = [ ... ]
              value:NX_FLOAT64[31] = [ ... ]
      detectors:NXnote
        I0_USAXS:NXdetector
          data --> /entry/instrument/bluesky/streams/primary/I0_USAXS
      monochromator:NXmonochromator
        energy --> /entry/instrument/bluesky/streams/baseline/monochromator_dcm_energy/
↪value_start
        feedback_on --> /entry/instrument/bluesky/streams/baseline/monochromator_
↪feedback_on/value_start
        mode --> /entry/instrument/bluesky/streams/baseline/monochromator_dcm_mode/value_
↪start
        theta --> /entry/instrument/bluesky/streams/baseline/monochromator_dcm_theta/
↪value_start
        wavelength --> /entry/instrument/bluesky/streams/baseline/monochromator_dcm_
↪wavelength/value_start
        y_offset --> /entry/instrument/bluesky/streams/baseline/monochromator_dcm_y_
↪offset/value_start
      positioners:NXnote
        m_stage_r:NXpositioner
          value --> /entry/instrument/bluesky/streams/primary/m_stage_r
      source:NXsource
        name:NX_CHAR = Bluesky framework
        probe:NX_CHAR = x-ray
        type:NX_CHAR = Synchrotron X-ray Source
```

APS-specific HDF5/NeXus File Structures

Examples of additional structure in NeXus file added by *NXWriterAPS()*:

```
  source:NXsource
    @NX_class = NXsource
    @target = /entry/instrument/source
    current --> /entry/instrument/bluesky/streams/baseline/aps_current/value_start
    energy:NX_INT64[] =
      @units = GeV
    fill_number --> /entry/instrument/bluesky/streams/baseline/aps_fill_number/value_
↪start
    name:NX_CHAR = Advanced Photon Source
      @short_name = APS
    probe:NX_CHAR = x-ray
    type:NX_CHAR = Synchrotron X-ray Source
  undulator:NXinsertion_device
    @NX_class = NXinsertion_device
    @target = /entry/instrument/undulator
    device --> /entry/instrument/bluesky/streams/baseline/undulator_downstream_device/
↪value_start
    energy --> /entry/instrument/bluesky/streams/baseline/undulator_downstream_energy/
↪value_start
    energy_taper --> /entry/instrument/bluesky/streams/baseline/undulator_downstream_
↪energy_taper/value_start
    gap --> /entry/instrument/bluesky/streams/baseline/undulator_downstream_gap/value_
↪start
    gap_taper --> /entry/instrument/bluesky/streams/baseline/undulator_downstream_gap_
↪taper/value_start
    harmonic_value --> /entry/instrument/bluesky/streams/baseline/undulator_downstream_
↪harmonic_value/value_start
    location --> /entry/instrument/bluesky/streams/baseline/undulator_downstream_
↪location/value_start
    total_power --> /entry/instrument/bluesky/streams/baseline/undulator_downstream_
↪total_power/value_start
    type:NX_CHAR = undulator
    version --> /entry/instrument/bluesky/streams/baseline/undulator_downstream_version/
↪value_start
```

SPEC File Structure

EXAMPLE : use as Bluesky callback:

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

EXAMPLE : use as writer from Databroker:

```
from apstools import SpecWriterCallback
specwriter = SpecWriterCallback()
for key, doc in db.get_documents(db["a123456"]):
```

(continues on next page)

(continued from previous page)

```
    specwriter.receiver(key, doc)
print("Look at SPEC data file: "+specwriter.spec_filename)
```

EXAMPLE : use as writer from Databroker with customizations:

```
from apstools import SpecWriterCallback

# write into file: /tmp/cerium.spec
specwriter = SpecWriterCallback(filename="/tmp/cerium.spec")
for key, doc in db.get_documents(db["abcd123"]):
    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)
```

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
```

(continues on next page)

(continued from previous page)

```
-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
-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
```

Source Code

Base Class for File Writer Callbacks

FileWriterCallbackBase(*args, **kwargs)	Base class for filewriter callbacks.

class apstools.callbacks.callback_base.**FileWriterCallbackBase**(**args*, ***kwargs*)

Base class for filewriter callbacks.

New with apstools release 1.3.0.

Applications should subclass and rewrite the `writer()` method.

The local buffers are cleared when a start document is received. Content is collected here from each document until the stop document. The content is written once the stop document is received.

User Interface methods

receiver(key, doc)	bluesky callback (handles a stream of documents)

Internal methods

clear()	delete any saved data from the cache and reinitialize
make_file_name()	generate a file name to be used as default
writer()	print summary of run as diagnostic

Document Handler methods

bulk_events(doc)	Deprecated.
datum(doc)	Like an event, but for data recorded outside of bluesky.
descriptor(doc)	description of the data stream to be acquired
event(doc)	a single "row" of data
resource(doc)	like a descriptor, but for data recorded outside of bluesky
start(doc)	beginning of a run, clear cache and collect metadata

continues on next page

Table 44 – continued from previous page

stop(doc)	end of the run, end collection and initiate the `writer()` method

bulk_events(*doc*)
Deprecated. Use EventPage instead.

clear()
delete any saved data from the cache and reinitialize

datum(*doc*)
Like an event, but for data recorded outside of bluesky.

Example:

```
Datum
=====
datum_id        : 621caa0f-70f1-4e3d-8718-b5123d434502/0
datum_kwargs    :
HDF5_file_name  : /mnt/usaxscontrol/USAXS_data/2020-06/06_10_Minjee_waxs/
↪AGIX3N1_0699.hdf
point_number    : 0
resource        : 621caa0f-70f1-4e3d-8718-b5123d434502
```

descriptor(*doc*)
description of the data stream to be acquired

event(*doc*)
a single "row" of data

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

default format: {ymd}-{hms}-S{scan_id}-{short_uid}.{ext} where the time (the run start time):

- ymd = {year:4d}{month:02d}{day:02d}
- hms = {hour:02d}{minute:02d}{second:02d}

override in subclass to change

receiver(*key*, *doc*)
bluesky callback (handles a stream of documents)

resource(*doc*)
like a descriptor, but for data recorded outside of bluesky

start(*doc*)
beginning of a run, clear cache and collect metadata

stop(*doc*)
end of the run, end collection and initiate the `writer()` method

writer()
print summary of run as diagnostic

override this method in subclass to write a file

NeXus File Writer Callbacks

NXWriter(*args, **kwargs)	General class for writing HDF5/NeXus file (using only NeXus base classes).
NXWriterAPS(*args, **kwargs)	Customize *NXWriter* with APS-specific content.

class apstools.callbacks.nexus_writer.**NXWriter**(**args*, ***kwargs*)

General class for writing HDF5/NeXus file (using only NeXus base classes).

New with apstools release 1.3.0.

One scan is written to one HDF5/NeXus file.

METHODS

writer()	write collected data to HDF5/NeXus data file
h5string(text)	Format string for h5py interface.
add_dataset_attributes(ds, v[, long_name])	add attributes from v dictionary to dataset ds
assign_signal_type()	decide if a signal in the primary stream is a detector or a positioner
create_NX_group(parent, specification)	create an h5 group with named NeXus class (specification)
get_sample_title()	return the title for this sample
get_stream_link(signal[, stream, ref])	return the h5 object for `signal`
write_data(parent)	group: /entry/data:NXdata
write_detector(parent)	group: /entry/instrument/detectors:NXnote/DETECTOR:NXdetector
write_entry()	group: /entry/data:NXentry
write_instrument(parent)	group: /entry/instrument:NXinstrument
write_metadata(parent)	group: /entry/instrument/bluesky/metadata:NXnote
write_monochromator(parent)	group: /entry/instrument/monochromator:NXmonochromator
write_positioner(parent)	group: /entry/instrument/positioners:NXnote/POSITIONER:NXpositioner
write_root(filename)	root of the HDF5 file
write_sample(parent)	group: /entry/sample:NXsample
write_slits(parent)	group: /entry/instrument/slits:NXnote/SLIT:NXslit
write_source(parent)	group: /entry/instrument/source:NXsource
write_streams(parent)	group: /entry/instrument/bluesky/streams:NXnote
write_user(parent)	group: /entry/contact:NXuser

add_dataset_attributes(*ds*, *v*, *long_name=None*)

add attributes from v dictionary to dataset ds

assign_signal_type()

decide if a signal in the primary stream is a detector or a positioner

create_NX_group(*parent*, *specification*)

create an h5 group with named NeXus class (specification)

getResourceFile(*resource_id*)

full path to the resource file specified by uid `resource_id`

override in subclass as needed

get_sample_title()

return the title for this sample

default title: {plan_name}-S{scan_id}-{short_uid}

get_stream_link(*signal*, *stream=None*, *ref=None*)
return the h5 object for `signal`

DEFAULTS

`stream` : `baseline` `key` : `value_start`

h5string(*text*)
Format string for h5py interface.

write_data(*parent*)
group: /entry/data:NXdata

write_detector(*parent*)
group: /entry/instrument/detectors:NXnote/DETECTOR:NXdetector

write_entry()
group: /entry/data:NXentry

write_instrument(*parent*)
group: /entry/instrument:NXinstrument

write_metadata(*parent*)
group: /entry/instrument/bluesky/metadata:NXnote

metadata from the bluesky start document

write_monochromator(*parent*)
group: /entry/instrument/monochromator:NXmonochromator

write_positioner(*parent*)
group: /entry/instrument/positioners:NXnote/POSITIONER:NXpositioner

write_root(*filename*)
root of the HDF5 file

write_sample(*parent*)
group: /entry/sample:NXsample

write_slits(*parent*)
group: /entry/instrument/slits:NXnote/SLIT:NXslit

override in subclass to store content, name patterns vary with each instrument

write_source(*parent*)
group: /entry/instrument/source:NXsource

Note: this is (somewhat) generic, override for a different source

write_streams(*parent*)
group: /entry/instrument/bluesky/streams:NXnote

data from all the bluesky streams

write_user(*parent*)
group: /entry/contact:NXuser

writer()
write collected data to HDF5/NeXus data file

class apstools.callbacks.nexus_writer.**NXWriterAPS**(**args*, ***kwargs*)
Customize *NXWriter* with APS-specific content.

New with apstools release 1.3.0.

- Adds */entry/instrument/undulator* group if metadata exists.

- Adds APS information to */entry/instrument/source* group.

write_instrument(parent)	group: /entry/instrument:NXinstrument
write_source(parent)	group: /entry/instrument/source:NXsource
write_undulator(parent)	group: /entry/instrument/undulator:NXinsertion_device

write_instrument(*parent*)
group: /entry/instrument:NXinstrument

write_source(*parent*)
group: /entry/instrument/source:NXsource

Note: this is specific to the APS, override for a different source

write_undulator(*parent*)
group: /entry/instrument/undulator:NXinsertion_device

SPEC Data File Writer Callback

EXAMPLE:

Execution of this plan (with `RE(myPlan())`):

```
def myPlan():
    yield from bps.open_run()
    spec_comment("this is a start document comment", "start")
    spec_comment("this is a descriptor document comment", "descriptor")
    yield bps.Msg('checkpoint')
    yield from bps.trigger_and_read([scaler])
    spec_comment("this is an event document comment after the first read")
    yield from bps.sleep(2)
    yield bps.Msg('checkpoint')
    yield from bps.trigger_and_read([scaler])
    spec_comment("this is an event document comment after the second read")
    spec_comment("this is a stop document comment", "stop")
    yield from bps.close_run()
```

results in this SPEC file output:

```
#S 1145  myPlan()
#D Mon Jan 28 12:48:09 2019
#C Mon Jan 28 12:48:09 2019.  plan_type = generator
#C Mon Jan 28 12:48:09 2019.  uid = ef98648a-8e3a-4e7e-ac99-3290c9b5fca7
#C Mon Jan 28 12:48:09 2019.  this is a start document comment
#C Mon Jan 28 12:48:09 2019.  this is a descriptor document comment
#MD APSTOOLS_VERSION = 2019.0103.0+5.g0f4e8b2
#MD BLUESKY_VERSION = 1.4.1
#MD OPHYD_VERSION = 1.3.0
#MD SESSION_START = 2019-01-28 12:19:25.446836
#MD beamline_id = developer
#MD ipython_session_start = 2018-02-14 12:54:06.447450
#MD login_id = mintadmin@mint-vm
#MD pid = 21784
#MD proposal_id = None
```

(continues on next page)

(continued from previous page)

```
#N 2
#L Epoch_float  scaler_time  Epoch
1.4297869205474854 1.1 1
4.596935987472534 1.1 5
#C Mon Jan 28 12:48:11 2019.  this is an event document comment after the first read
#C Mon Jan 28 12:48:14 2019.  this is an event document comment after the second read
#C Mon Jan 28 12:48:14 2019.  this is a stop document comment
#C Mon Jan 28 12:48:14 2019.  num_events_primary = 2
#C Mon Jan 28 12:48:14 2019.  exit_status = success
```

SpecWriterCallback([filename, auto_write, ...])	Collect data from Bluesky RunEngine documents to write as SPEC data.
spec_comment(comment[, doc, writer])	make it easy to add spec-style comments in a custom plan

class apstools.callbacks.spec_file_writer.**SpecWriterCallback**(*filename=None*, *auto_write=True*, *RE=None*, *reset_scan_id=False*)

Collect data from Bluesky RunEngine documents to write as SPEC data.

This gathers data from all documents in a scan and appends scan to the file when the `stop` document is received. One or more scans can be written to the same file. The file format is text.

Note: `SpecWriterCallback()` does **not** inherit from `FileWriterCallbackBase()`.

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.

RE *object* : Instance of `bluesky.RunEngine` or `None`.

reset_scan_id *boolean* : (optional) If True, and filename exists, then sets `RE.md.scan_id` to highest scan number in existing SPEC data file. default: False

User Interface methods

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

continues on next page

Table 50 – continued from previous page

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*, *scan_id=None*, *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

apstools.callbacks.spec_file_writer.**spec_comment**(*comment*, *doc=None*, *writer=None*)
make it easy to add spec-style comments in a custom plan

These comments *only* go into the SPEC data file.

PARAMETERS

comment *string* : (optional) Comment text to be written. SPEC expects it to be only one line!

doc *string* : (optional) Bluesky RunEngine document type. One of: `start descriptor event resource datum stop` (default: `event`)

writer *obj* : (optional) Instance of `SpecWriterCallback()`, typically: `specwriter = SpecWriterCallback()`

2.4.4 Plans

Plans that might be useful at the APS when using Bluesky.

Plans and Support by Activity

Batch Scanning

execute_command_list(filename, commands[, md])	plan: execute the command list
get_command_list(filename)	return command list from either text or Excel file
parse_Excel_command_file(filename)	parse an Excel spreadsheet with commands, return as command list
parse_text_command_file(filename)	parse a text file with commands, return as command list
register_command_handler([handler])	Define the function called to execute the command list
run_command_file(filename[, md])	plan: execute a list of commands from a text or Excel file
summarize_command_file(filename)	print the command list from a text or Excel file

Custom Scans

documentation_run(text[, stream, bec, md])	Save text as a bluesky run.
lineup(counter, axis, minus, plus, npts[, ...])	lineup and center a given axis, relative to current position
nscan(detectors, *motor_sets[, num, ...])	Scan over `n` variables moved together, each in equally spaced steps.
snapshot(obj_list[, stream, md])	bluesky plan: record current values of list of ophyd signals
sscan_1D(sscan[, poll_delay_s, ...])	simple 1-D scan using EPICS synApps sscan record
TuneAxis(signals, axis[, signal_name])	tune an axis with a signal
tune_axes(axes)	Bluesky plan to tune a list of axes in sequence

Overall

addDeviceDataAsStream(devices, label)	add an ophyd Device as an additional document stream
command_list_as_table(commands[, show_raw])	format a command list as a pyRestTable.Table object
documentation_run(text[, stream, bec, md])	Save text as a bluesky run.
execute_command_list(filename, commands[, md])	plan: execute the command list
get_command_list(filename)	return command list from either text or Excel file
lineup(counter, axis, minus, plus, npts[, ...])	lineup and center a given axis, relative to current position
nscan(detectors, *motor_sets[, num, ...])	Scan over n variables moved together, each in equally spaced steps.
parse_Excel_command_file(filename)	parse an Excel spreadsheet with commands, return as command list
parse_text_command_file(filename)	parse a text file with commands, return as command list
register_command_handler([handler])	Define the function called to execute the command list
run_command_file(filename[, md])	plan: execute a list of commands from a text or Excel file
snapshot(obj_list[, stream, md])	bluesky plan: record current values of list of ophyd signals
sscan_1D(sscan[, poll_delay_s, ...])	simple 1-D scan using EPICS synApps sscan record
summarize_command_file(filename)	print the command list from a text or Excel file
TuneAxis(signals, axis[, signal_name])	tune an axis with a signal
tune_axes(axes)	Bluesky plan to tune a list of axes in sequence

Also consult the Index under the *Bluesky* heading for links to the Callbacks, Devices, Exceptions, and Plans described here.

Submodules

Alignment plans

lineup(counter, axis, minus, plus, npts[, ...])	lineup and center a given axis, relative to current position
tune_axes(axes)	Bluesky plan to tune a list of axes in sequence
TuneAxis(signals, axis[, signal_name])	tune an axis with a signal
TuneResults(*args, **kwargs)	Provides bps.read() as a Device

class apstools.plans.alignment.**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: the TuneAxis() class*.

tune([width, num, peak_factor, 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([peak_factor])	returns True if a peak was detected, otherwise False

SEE ALSO

tune_axes(axes)	Bluesky plan to tune a list of axes in sequence

multi_pass_tune(*width=None*, *step_factor=None*, *num=None*, *pass_max=None*, *peak_factor=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 4)

peak_factor *float* : peak maximum must be greater than `peak_factor*minimum` (default: 4)

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

md *dict* : (optional) metadata

peak_detected(*peak_factor=None*)

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*, *peak_factor=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

class apstools.plans.alignment.**TuneResults**(**args: Any*, ***kwargs: Any*)
Provides bps.read() as a Device

apstools.plans.alignment.**lineup**(*counter*, *axis*, *minus*, *plus*, *npts*, *time_s=0.1*, *peak_factor=4*, *width_factor=0.8*, *md=None*)
lineup and center a given axis, relative to current position

PARAMETERS

counter *object* : instance of ophyd.Signal (or subclass such as ophyd.scaler.ScalerChannel) dependent measurement to be maximized

axis movable *object* : instance of ophyd.Signal (or subclass such as EpicsMotor) independent axis to use for alignment

minus *float* : first point of scan at this offset from starting position

plus *float* : last point of scan at this offset from starting position

npts *int* : number of data points in the scan

time_s *float* : count time per step (if counter is ScalerChannel object) (default: 0.1)

peak_factor *float* : peak maximum must be greater than `peak_factor*minimum` (default: 4)

width_factor *float* : fwhm must be less than `width_factor*plot_range` (default: 0.8)

EXAMPLE:

```
RE(lineup(diode, foemirror.theta, -30, 30, 30, 1.0))
```

apstools.plans.alignment.**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])
```

SEE ALSO

TuneAxis(signals, axis[, signal_name])	tune an axis with a signal

nscan plan

CommandFileReadError	Exception when reading a command file.
command_list_as_table(commands[, show_raw])	format a command list as a pyRestTable.Table object
execute_command_list(filename, commands[, md])	plan: execute the command list
get_command_list(filename)	return command list from either text or Excel file
parse_Excel_command_file(filename)	parse an Excel spreadsheet with commands, return as command list
parse_text_command_file(filename)	parse a text file with commands, return as command list
register_command_handler([handler])	Define the function called to execute the command list

continues on next page

Table 58 – continued from previous page

run_command_file(filename[, md])	plan: execute a list of commands from a text or Excel file
summarize_command_file(filename)	print the command list from a text or Excel file

exception apstools.plans.command_list.**CommandFileReadError**
Exception when reading a command file.

apstools.plans.command_list.**command_list_as_table**(*commands*, *show_raw=False*)
format a command list as a pyRestTable.Table object

apstools.plans.command_list.**execute_command_list**(*filename*, *commands*, *md=None*)
plan: execute the command list

The command list is a tuple described below.

- Only recognized commands will be executed.
- Unrecognized commands will be reported as comments.

See example implementation with APS USAXS instrument: https://github.com/APS-USAXS/ipython-usaxs/blob/5db882c47d935c593968f1e2144d35bec7d0181e/profile_bluesky/startup/50-plans.py#L381-L469

PARAMETERS

filename *str* : Name of input text file. Can be relative or absolute path, such as "actions.txt", "../sample.txt", or "/path/to/overnight.txt".

commands *[command]* : List of command tuples for use in `execute_command_list()`

where

command *tuple* : (action, OrderedDict, line_number, raw_command)

action *str* : names a known action to be handled

parameters *list* : List of parameters for the action. The list is empty of there are no values

line_number *int* : line number (1-based) from the input text file

raw_command *str* or *[str]* : contents from input file, such as: `SAXS 0 0 0 blank`

SEE ALSO

execute_command_list(filename, commands[, md])	plan: execute the command list
register_command_handler([handler])	Define the function called to execute the command list
run_command_file(filename[, md])	plan: execute a list of commands from a text or Excel file
summarize_command_file(filename)	print the command list from a text or Excel file
parse_Excel_command_file(filename)	parse an Excel spreadsheet with commands, return as command list
parse_text_command_file(filename)	parse a text file with commands, return as command list

new in apstools release 1.1.7

apstools.plans.command_list.**get_command_list**(*filename*)
return command list from either text or Excel file

SEE ALSO

execute_command_list(filename, commands[, md])	plan: execute the command list
get_command_list(filename)	return command list from either text or Excel file
register_command_handler([handler])	Define the function called to execute the command list
run_command_file(filename[, md])	plan: execute a list of commands from a text or Excel file
summarize_command_file(filename)	print the command list from a text or Excel file
parse_Excel_command_file(filename)	parse an Excel spreadsheet with commands, return as command list
parse_text_command_file(filename)	parse a text file with commands, return as command list

new in apstools release 1.1.7

apstools.plans.command_list.**parse_Excel_command_file**(*filename*)

parse an Excel spreadsheet with commands, return as command list

TEXT view of spreadsheet (Excel file line numbers shown):

```
[1] List of sample scans to be run
[2]
[3]
[4] scan    sx  sy  thickness   sample name
[5] FlyScan 0   0   0   blank
[6] FlyScan 5   2   0   blank
```

PARAMETERS

filename *str* : Name of input Excel spreadsheet file. Can be relative or absolute path, such as "actions.xslx", "../sample.xslx", or "/path/to/overnight.xslx".

RETURNS

list of commands *[command]* : List of command tuples for use in `execute_command_list()`

RAISES

FileNotFoundError if file cannot be found

SEE ALSO

get_command_list(filename)	return command list from either text or Excel file
register_command_handler([handler])	Define the function called to execute the command list
run_command_file(filename[, md])	plan: execute a list of commands from a text or Excel file
summarize_command_file(filename)	print the command list from a text or Excel file
parse_text_command_file(filename)	parse a text file with commands, return as command list

new in apstools release 1.1.7

apstools.plans.command_list.**parse_text_command_file**(*filename*)

parse a text file with commands, return as command list

- The text file is interpreted line-by-line.

- Blank lines are ignored.
- A pound sign (#) marks the rest of that line as a comment.
- All remaining lines are interpreted as commands with arguments.

Example of text file (no line numbers shown):

```
#List of sample scans to be run
# pound sign starts a comment (through end of line)

# action  value
mono_shutter open

# action  x y width height
uslits 0 0 0.4 1.2

# action  sx  sy  thickness   sample name
FlyScan 0   0   0   blank
FlyScan 5   2   0   "empty container"

# action  sx  sy  thickness   sample name
SAXS 0 0 0 blank

# action  value
mono_shutter close
```

PARAMETERS

filename *str* : Name of input text file. Can be relative or absolute path, such as "actions.txt", "../sample.txt", or "/path/to/overnight.txt".

RETURNS

list of commands *[command]* : List of command tuples for use in `execute_command_list()`

RAISES

FileNotFoundError if file cannot be found

SEE ALSO

execute_command_list(filename, commands[, md])	plan: execute the command list
get_command_list(filename)	return command list from either text or Excel file
register_command_handler([handler])	Define the function called to execute the command list
run_command_file(filename[, md])	plan: execute a list of commands from a text or Excel file
summarize_command_file(filename)	print the command list from a text or Excel file
parse_Excel_command_file(filename)	parse an Excel spreadsheet with commands, return as command list

new in apstools release 1.1.7

apstools.plans.command_list.**register_command_handler**(*handler=None*)

Define the function called to execute the command list

PARAMETERS

handler *obj* : Reference of the `execute_command_list` function to be used from `run_command_file()`. If `None` or not provided, will reset to `execute_command_list()`, which is also the initial setting.

SEE ALSO

execute_command_list(filename, commands[, md])	plan: execute the command list
get_command_list(filename)	return command list from either text or Excel file
register_command_handler([handler])	Define the function called to execute the command list
summarize_command_file(filename)	print the command list from a text or Excel file
parse_Excel_command_file(filename)	parse an Excel spreadsheet with commands, return as command list
parse_text_command_file(filename)	parse a text file with commands, return as command list

new in apstools release 1.1.7

apstools.plans.command_list.**run_command_file**(*filename*, *md=None*)

plan: execute a list of commands from a text or Excel file

- Parse the file into a command list
- yield the command list to the RunEngine (or other)

SEE ALSO

execute_command_list(filename, commands[, md])	plan: execute the command list
get_command_list(filename)	return command list from either text or Excel file
register_command_handler([handler])	Define the function called to execute the command list
summarize_command_file(filename)	print the command list from a text or Excel file
parse_Excel_command_file(filename)	parse an Excel spreadsheet with commands, return as command list
parse_text_command_file(filename)	parse a text file with commands, return as command list

new in apstools release 1.1.7

apstools.plans.command_list.**summarize_command_file**(*filename*)

print the command list from a text or Excel file

SEE ALSO

execute_command_list(filename, commands[, md])	plan: execute the command list
get_command_list(filename)	return command list from either text or Excel file
run_command_file(filename[, md])	plan: execute a list of commands from a text or Excel file
parse_Excel_command_file(filename)	parse an Excel spreadsheet with commands, return as command list
parse_text_command_file(filename)	parse a text file with commands, return as command list

new in apstools release 1.1.7

Documentation of batch runs

addDeviceDataAsStream(devices, label)	add an ophyd Device as an additional document stream
documentation_run(text[, stream, bec, md])	Save text as a bluesky run.

apstools.plans.doc_run.**addDeviceDataAsStream**(*devices*, *label*)
add an ophyd Device as an additional document stream

Use this within a custom plan, such as this example:

```
from apstools.plans import addDeviceStream
...
yield from bps.open_run()
# ...
yield from addDeviceDataAsStream(prescanDeviceList, "metadata_prescan")
# ...
yield from custom_scan_procedure()
# ...
yield from addDeviceDataAsStream(postscanDeviceList, "metadata_postscan")
# ...
yield from bps.close_run()
```

apstools.plans.doc_run.**documentation_run**(*text*, *stream=None*, *bec=None*, *md=None*)
Save text as a bluesky run.

PARAMETERS

text *str* : Text to be written.

stream *str* : document stream, default: "primary"

bec *object* : Instance of *bluesky.BestEffortCallback*, default: get from IPython shell

md *dict* (optional): metadata dictionary

nscan plan

nscan(detectors, *motor_sets[, num, ...])	Scan over n variables moved together, each in equally spaced steps.

apstools.plans.nscan_support.**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/apstools/blob/master/docs/source/resources/demo_nscan.ipynb

snapshot Support

snapshot(obj_list[, stream, md])	bluesky plan: record current values of list of ophyd signals

apstools.plans.snapshot_support.**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

sscan Record plans

sscan_1D(sscan[, poll_delay_s, ...])	simple 1-D scan using EPICS synApps sscan record

apstools.plans.sscan_support.**sscan_1D**(*sscan*, *poll_delay_s=0.001*, *phase_timeout_s=60.0*, *running_stream='primary'*, *final_array_stream=None*, *device_settings_stream='settings'*, *md=None*)

simple 1-D scan using EPICS synApps sscan record

assumes the sscan record has already been setup properly for a scan

PARAMETERS

sscan ***Device*** **:** one EPICS sscan record (instance of *apstools.synApps.sscanRecord*)

running_stream ***str*** [or *None*] (default: `"primary"`) Name of document stream to write positioners and detectors data made available while the sscan is running. This is typically the scan data, row by row. If set to *None*, this stream will not be written.

final_array_stream ***str*** **or** `None` **:** Name of document stream to write positioners and detectors data posted *after* the sscan has ended. If set to *None*, this stream will not be written. (default: `None`)

device_settings_stream ***str*** **or** `None` **:** Name of document stream to write *settings* of the sscan device. This is all the information returned by `sscan.read()`. If set to *None*, this stream will not be written. (default: `"settings"`)

poll_delay_s ***float*** **:** How long to sleep during each polling loop while collecting interim data values and waiting for sscan to complete. Must be a number between zero and 0.1 seconds. (default: 0.001 seconds)

phase_timeout_s ***float*** **:** How long to wait after last update of the `sscan.FAZE`. When scanning, we expect the scan phase to update regularly as positioners move and detectors are triggered. If the scan hangs for some

reason, this is a way to end the plan early. To cancel this feature, set it to `None`. (default: 60 seconds)

NOTE about the document stream names

Make certain the names for the document streams are different from each other. If you make them all the same (such as `primary`), you will have difficulty when reading your data later on.

Don't cross the streams!

EXAMPLE

Assume that the chosen sscan record has already been setup.

from apstools.devices import sscanDevice scans = sscanDevice(P, name="scans")

from apstools.plans import sscan_1D RE(sscan_1D(scans.scan1), md=dict(purpose="demo"))

2.4.5 Utilities

Various utilities to help APS use the Bluesky framework.

Also consult the Index under the *apstools* heading for links to the Exceptions, and Utilities described here.

Utilities by Activity

Finding

findbyname(oname[, force_rebuild, ns])	Find the ophyd (dotted name) object associated with the given ophyd name.
findbypv(pvname[, force_rebuild, ns])	Find all ophyd objects associated with the given EPICS PV.
`findCatalogsInNamespace`()	

Listing

listdevice(obj[, scope, cname, dname, ...])	Describe the signal information from device `obj` in a pandas DataFrame.
listobjects([show_pv, printing, verbose, ...])	Show all the ophyd Signal and Device objects defined as globals.
listplans([base, trunc])	List all plans.
listRunKeys(scan_id[, key_fragment, db, ...])	Convenience function to list all keys (column names) in the scan's stream (default: primary).
ListRuns([cat, query, keys, missing, num, ...])	List the runs from the given catalog according to some options.
listruns([cat, keys, missing, num, ...])	List runs from catalog.

Reporting

print_RE_md([dictionary, fmt, printing])	custom print the RunEngine metadata in a table
print_snapshot_list(db[, printing])	print (stdout) a list of all snapshots in the databroker
SlitGeometry(width, height, x, y)	Slit size and center as a named tuple

Other Utilities

cleanupText(text)	convert text so it can be used as a dictionary key
connect_pvlist(pvlist[, wait, timeout, ...])	Given list of EPICS PV names, return dict of EpicsSignal objects.
EmailNotifications([sender])	send email notifications when requested
select_live_plot(bec, signal)	Get the *first* live plot that matches `signal`.
select_mpl_figure(x, y)	Get the MatPlotLib Figure window for y vs x.
trim_plot_by_name([n, plots])	Find the plot(s) by name and replot with at most the last *n* lines.
trim_plot_lines(bec, n, x, y)	Find the plot with axes x and y and replot with at most the last *n* lines.
trim_string_for_EPICS(msg)	String must not exceed EPICS PV length.
unix(command[, raises])	Run a UNIX command, returns (stdout, stderr).

General

cleanupText(text)	convert text so it can be used as a dictionary key
command_list_as_table(commands[, show_raw])	format a command list as a pyRestTable.Table object
connect_pvlist(pvlist[, wait, timeout, ...])	Given list of EPICS PV names, return dict of EpicsSignal objects.
copy_filtered_catalog(source_cat, target_cat)	copy filtered runs from source_cat to target_cat
db_query(db, query)	Searches the databroker v2 database.
dictionary_table(dictionary, **kwargs)	return a text table from `dictionary`
EmailNotifications([sender])	send email notifications when requested
ExcelDatabaseFileBase([ignore_extra])	base class: read-only support for Excel files, treat them like databases
ExcelDatabaseFileGeneric(filename[, ...])	Generic (read-only) handling of Excel spreadsheet-as-database
ExcelReadError(*args, **kwargs)	Exception when reading Excel spreadsheet.
findbyname(oname[, force_rebuild, ns])	Find the ophyd (dotted name) object associated with the given ophyd name.
findbypv(pvname[, force_rebuild, ns])	Find all ophyd objects associated with the given EPICS PV.
findCatalogsInNamespace()	
full_dotted_name(obj)	Return the full dotted name
getCatalog([ref])	
getDatabase([db, catalog_name])	Return Bluesky database using keyword guides or default choice.

continues on next page

Table 74 – continued from previous page

`getDefaultCatalog`()	
getDefaultDatabase()	Find the "default" database (has the most recent run).
getDefaultNamespace([attr])	get the IPython shell's namespace dictionary (or globals() if not found)
getRunData(scan_id[, db, stream, query, use_v1])	Convenience function to get the run's data.
getRunDataValue(scan_id, key[, db, stream, ...])	Convenience function to get value of key in run stream.
getStreamValues(scan_id[, key_fragment, db, ...])	Get values from a previous scan stream in a databroker catalog.
ipython_profile_name()	return the name of the current ipython profile or `None`
itemizer(fmt, items)	Format a list of items.
listdevice(obj[, scope, cname, dname, ...])	Describe the signal information from device `obj` in a pandas DataFrame.
listobjects([show_pv, printing, verbose, ...])	Show all the ophyd Signal and Device objects defined as globals.
listplans([base, trunc])	List all plans.
listRunKeys(scan_id[, key_fragment, db, ...])	Convenience function to list all keys (column names) in the scan's stream (default: primary).
ListRuns([cat, query, keys, missing, num, ...])	List the runs from the given catalog according to some options.
listruns([cat, keys, missing, num, ...])	List runs from catalog.
OverrideParameters()	Define parameters that can be overridden from a user configuration file.
pairwise(iterable)	break a list (or other iterable) into pairs
print_RE_md([dictionary, fmt, printing])	custom print the RunEngine metadata in a table
print_snapshot_list(db[, printing])	print (stdout) a list of all snapshots in the databroker
quantify_md_key_use([key, db, catalog_name, ...])	Print table of different `key` values and how many times each appears.
redefine_motor_position(motor, new_position)	Set EPICS motor record's user coordinate to `new_position`.
replay(headers[, callback, sort])	Replay the document stream from one (or more) scans (headers).
rss_mem()	return memory used by this process
run_in_thread(func)	(decorator) run `func` in thread
safe_ophyd_name(text)	make text safe to be used as an ophyd object name
select_live_plot(bec, signal)	Get the *first* live plot that matches `signal`.
select_mpl_figure(x, y)	Get the MatPlotLib Figure window for y vs x.
split_quoted_line(line)	splits a line into words some of which might be quoted
summarize_runs([since, db])	Report bluesky run metrics from the databroker.
text_encode(source)	Encode `source` using the default codepoint.
to_unicode_or_bust(obj[, encoding])	from: http://farmdev.com/talks/unicode/ .
trim_plot_by_name([n, plots])	Find the plot(s) by name and replot with at most the last *n* lines.
trim_plot_lines(bec, n, x, y)	Find the plot with axes x and y and replot with at most the last *n* lines.
trim_string_for_EPICS(msg)	String must not exceed EPICS PV length.
unix(command[, raises])	Run a UNIX command, returns (stdout, stderr).

Submodules

Working with databroker catalogs

copy_filtered_catalog(source_cat, target_cat)	copy filtered runs from source_cat to target_cat
findCatalogsInNamespace()	
getCatalog([ref])	
getDatabase([db, catalog_name])	Return Bluesky database using keyword guides or default choice.
getDefaultCatalog()	
getDefaultDatabase()	Find the "default" database (has the most recent run).
getStreamValues(scan_id[, key_fragment, db, ...])	Get values from a previous scan stream in a databroker catalog.
quantify_md_key_use([key, db, catalog_name, ...])	Print table of different key values and how many times each appears.

apstools.utils.catalog.**copy_filtered_catalog**(*source_cat*, *target_cat*, *query=None*)

copy filtered runs from source_cat to target_cat

PARAMETERS

source_cat *obj* : instance of *databroker.Broker* or *databroker.catalog[name]*

target_cat *obj* : instance of *databroker.Broker* or *databroker.catalog[name]*

query *dict* : mongo query dictionary, used to filter the results (default: {})

see: https://docs.mongodb.com/manual/reference/operator/query/

example:

```
copy_filtered_catalog(
    databroker.Broker.named("mongodb_config"),
    databroker.catalog["test1"],
    {'plan_name': 'snapshot'})
```

apstools.utils.catalog.**getDatabase**(*db=None*, *catalog_name=None*)

Return Bluesky database using keyword guides or default choice.

PARAMETERS

db *object* : Bluesky database, an instance of databroker.catalog (default: see catalog_name keyword argument)

catalog_name *str* : Name of databroker v2 catalog, used when supplied db is None. (default: catalog with most recent run timestamp)

RETURNS

object or None: Bluesky database, an instance of databroker.catalog

(new in release 1.4.0)

apstools.utils.catalog.**getDefaultDatabase**()

Find the "default" database (has the most recent run).

Note that here, *database* and *catalog* mean the same.

This routine looks at all the database instances defined in the current session (console or notebook). If there is only one or no database instances defined as objects in the current session, the choice is simple. When there is more than one database instance in the current session, then the one with the most recent run timestamp is selected. In the case (as happens when starting with a new database) that the current database has **no** runs *and* another database instance is defined in the session *and* that additional database has runs in it (such as the previous database), then the database with the newest run timestamp (and not the newer empty database) will be chosen.

RETURNS

object or `None`: Bluesky database, an instance of `databroker.catalog`

(new in release 1.4.0)

`apstools.utils.catalog.`**`getStreamValues`**(*scan_id*, *key_fragment=''*, *db=None*, *stream='baseline'*, *query=None*, *use_v1=True*)

Get values from a previous scan stream in a databroker catalog.

Optionally, select only those data with names including `key_fragment`.

Tip: If the output is truncated, use `pd.set_option('display.max_rows', 300)` to increase the number of rows displayed.

PARAMETERS

scan_id *int* or *str* : Scan (run) identifier. Positive integer value is `scan_id` from run's metadata. Negative integer value is since most recent run in databroker. String is run's `uid` unique identifier (can abbreviate to the first characters needed to assure it is unique).

key_fragment *str* : Part or all of key name to be found in selected stream. For instance, if you specify `key_fragment="lakeshore"`, it will return all the keys that include `lakeshore`.

db *object* : Bluesky database, an instance of `databroker.catalog`. Default: will search existing session for instance.

stream *str* : Name of the bluesky data stream to obtain the data. Default: 'baseline'

query *dict* : mongo query dictionary, used to filter the results Default: `{}`

see: https://docs.mongodb.com/manual/reference/operator/query/

use_v1 *bool* : Chooses databroker API version between 'v1' or 'v2'. Default: `True` (meaning use the v1 API)

RETURNS

***object* :** pandas DataFrame with values from selected stream, search_string, and query

see: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

(new in apstools 1.5.1)

`apstools.utils.catalog.`**`quantify_md_key_use`**(*key=None*, *db=None*, *catalog_name=None*, *since=None*, *until=None*, *query=None*)

Print table of different `key` values and how many times each appears.

PARAMETERS

key *str* : one of the metadata keys in a run's start document (default: `plan_name`)

db *object* : Instance of databroker v1 `Broker` or v2 `catalog` (default: see `catalog_name` keyword argument)

catalog_name *str* : Name of databroker v2 catalog, used when supplied `db` is `None`. (default: `mongodb_config`)

since *str* **:** include runs that started on or after this ISO8601 time (default: `1995-01-01`)

until *str* **:** include runs that started before this ISO8601 time (default: `2100-12-31`)

query *dict* **:** mongo query dictionary, used to filter the results (default: `{}`)

see: https://docs.mongodb.com/manual/reference/operator/query/

EXAMPLES:

```
quantify_md_key_use(key="proposal_id")
quantify_md_key_use(key="plan_name", catalog_name="9idc", since="2020-07")
quantify_md_key_use(key="beamline_id", catalog_name="9idc")
quantify_md_key_use(key="beamline_id",
                    catalog_name="9idc",
                    query={'plan_name': 'Flyscan'},
                    since="2020",
                    until="2020-06-21 21:51")
quantify_md_key_use(catalog_name="8id", since="2020-01", until="2020-03")

In [8]: quantify_md_key_use(catalog_name="apstools_test")
========= =====
plan_name #runs
========= =====
count     26
scan      27
========= =====

In [9]: quantify_md_key_use(catalog_name="usaxs_test")
========================== =====
plan_name                  #runs
========================== =====
Flyscan                    1
TuneAxis.tune              1
count                      1
measure_USAXS_Transmission 1
run_Excel_file             1
snapshot                   1
tune_a2rp                  1
tune_ar                    1
tune_m2rp                  1
tune_mr                    1
========================== =====
```

Device information

listdevice(obj[, scope, cname, dname, ...])	Describe the signal information from device `obj` in a pandas DataFrame.

apstools.utils.device_info.**listdevice**(*obj*, *scope=None*, *cname=False*, *dname=True*, *show_pv=False*, *use_datetime=True*, *show_ancient=True*)

Describe the signal information from device `obj` in a pandas DataFrame.

Look through all subcomponents to find all the signals to be shown.

PARAMETERS

obj *object* : Instance of ophyd Signal or Device.

scope *str* or None : Scope of content to be shown.

- `"full"` (or `None`) shows all Signal components
- `"epics"` shows only EPICS-based Signals
- `"read"` shows only the signals returned by `obj.read()`

default: `None`

cname *bool* : Show the _control_ (Python, dotted) name in column `name`.

default: `False`

dname *bool* : Show the _data_ (databroker, with underlines) name in column `data name`.

default: `True`

show_pv *bool* : Show the EPICS process variable (PV) name in column `PV`.

default: `False`

use_datetime *bool* : Show the EPICS timestamp (time of last update) in column `timestamp`.

default: `True`

show_ancient *bool* : Show uninitialized EPICS process variables.

In EPICS, an uninitialized PV has a timestamp of 1990-01-01 UTC. This option enables or suppresses ancient values identified by timestamp from 1989. These are values only defined in the original `.db` file.

default: `True`

email Support

EmailNotifications([sender])	send email notifications when requested

class apstools.utils.email.**EmailNotifications**(*sender=None*)

send email notifications when requested

use default OS mail utility (so no credentials needed)

EXAMPLE

Send email(s) when *feedback_limits_approached* (a hypothetical boolean) is *True*:

```
# setup
from apstools.utils import EmailNotifications

SENDER_EMAIL = "instrument_user@email.host.tld"

email_notices = EmailNotifications(SENDER_EMAIL)
email_notices.add_addresses(
    # This list receives email when send() is called.
    "joe.user@goodmail.com",
    "instrument_team@email.host.tld",
    # others?
```

(continues on next page)

(continued from previous page)

```
)

# ... later

if feedback_limits_approached:
    # send emails to list
    subject = "Feedback problem"
    message = "Feedback is very close to its limits."
    email_notices.send(subject, message)
```

run_in_thread()
(decorator) run `func` in thread

USAGE:

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

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

Directory of the known plans

listplans([base, trunc])	List all plans.

apstools.utils.list_plans.**listplans**(*base=None*, *trunc=40*)
List all plans. (Actually, lists all generator functions).

NOTE: Can only detect generator functions. Bluesky plans are generator functions that generate `bluesky.Msg` objects. There is a PR to define a decorator that identifies a generator function as a bluesky plan.

PARAMETERS

base *object* or *None* : Object that contains plan methods (if `None`, use global namespace.) (default: `None`)

trunc *int* : Truncate long docstrings to no more than `trunc` characters. (default: 40)

Directory of bluesky runs

getRunData(scan_id[, db, stream, query, use_v1])	Convenience function to get the run's data.
getRunDataValue(scan_id, key[, db, stream, ...])	Convenience function to get value of key in run stream.
listRunKeys(scan_id[, key_fragment, db, ...])	Convenience function to list all keys (column names) in the scan's stream (default: primary).
ListRuns([cat, query, keys, missing, num, ...])	List the runs from the given catalog according to some options.
listruns([cat, keys, missing, num, ...])	List runs from catalog.
summarize_runs([since, db])	Report bluesky run metrics from the databroker.

class apstools.utils.list_runs.**ListRuns**(*cat: Optional[object] = None*, *query: Optional[dict] = None*, *keys: Optional[str] = None*, *missing: str = ''*, *num: int = 20*, *reverse: bool = True*, *since: Optional[str] = None*, *sortby: str = 'time'*, *timefmt: str = '%Y-%m-%d %H:%M:%S'*, *until: Optional[str] = None*, *ids: Optional[Any] = None*)

List the runs from the given catalog according to some options.

EXAMPLE:

```
ListRuns(cat).to_dataframe()
```

PUBLIC METHODS

to_dataframe()	Output as pandas DataFrame object
to_table([fmt])	Output as pyRestTable object.
parse_runs()	Parse the runs for the given metadata keys.

INTERNAL METHODS

_get_by_key(md, key)	Get run's metadata value by key.
_check_cat()	

_apply_search_filters()	Search for runs from the catalog.
_check_keys()	Check that self.keys is a list of strings.

parse_runs()
Parse the runs for the given metadata keys. Return a dict.

to_dataframe()
Output as pandas DataFrame object

to_table(*fmt=None*)
Output as pyRestTable object.

apstools.utils.list_runs.**getRunData**(*scan_id*, *db=None*, *stream='primary'*, *query=None*, *use_v1=True*)

Convenience function to get the run's data. Default is the `primary` stream.

PARAMETERS

scan_id *int* or *str* : Scan (run) identifier. Positive integer value is `scan_id` from run's metadata. Negative integer value is since most recent run in databroker. String is run's `uid` unique identifier (can abbreviate to the first characters needed to assure it is unique).

db *object* : Bluesky database, an instance of `databroker.catalog`. Default: will search existing session for instance.

stream *str* : Name of the bluesky data stream to obtain the data. Default: 'primary'

query *dict* : mongo query dictionary, used to filter the results Default: `{}`

see: https://docs.mongodb.com/manual/reference/operator/query/

use_v1 *bool* : Chooses databroker API version between 'v1' or 'v2'. Default: `True` (meaning use the v1 API)

(new in apstools 1.5.1)

apstools.utils.list_runs.**getRunDataValue**(*scan_id*, *key*, *db=None*, *stream='primary'*, *query=None*, *idx=-1*, *use_v1=True*)

Convenience function to get value of key in run stream.

Defaults are last value of key in primary stream.

PARAMETERS

scan_id *int* or *str* : Scan (run) identifier. Positive integer value is `scan_id` from run's metadata. Negative integer value is since most recent run in databroker. String is run's `uid` unique identifier (can abbreviate to the first characters needed to assure it is unique).

key *str* : Name of the key (data column) in the table of the stream's data. Must match *identically*.

db *object* : Bluesky database, an instance of `databroker.catalog`. Default: will search existing session for instance.

stream *str* : Name of the bluesky data stream to obtain the data. Default: 'primary'

query *dict* : mongo query dictionary, used to filter the results Default: `{}`

see: https://docs.mongodb.com/manual/reference/operator/query/

idx *int* or *str* : List index of value to be returned from column of table. Can be `0` for first value, `-1` for last value, `"mean"` for average value, or `"all"` for the full list of values. Default: `-1`

use_v1 *bool* : Chooses databroker API version between 'v1' or 'v2'. Default: `True` (meaning use the v1 API)

(new in apstools 1.5.1)

apstools.utils.list_runs.**listRunKeys**(*scan_id*, *key_fragment=''*, *db=None*, *stream='primary'*, *query=None*, *strict=False*, *use_v1=True*)

Convenience function to list all keys (column names) in the scan's stream (default: primary).

PARAMETERS

scan_id *int* or *str* : Scan (run) identifier. Positive integer value is `scan_id` from run's metadata. Negative integer value is since most recent run in databroker. String is run's `uid` unique identifier (can abbreviate to the first characters needed to assure it is unique).

key_fragment *str* : Part or all of key name to be found in selected stream. For instance, if you specify `key_fragment="lakeshore"`, it will return all the keys that include `lakeshore`.

db *object* : Bluesky database, an instance of `databroker.catalog`. Default: will search existing session for instance.

stream *str* : Name of the bluesky data stream to obtain the data. Default: 'primary'

query *dict* : mongo query dictionary, used to filter the results Default: `{}`

see: https://docs.mongodb.com/manual/reference/operator/query/

strict *bool* : Should the `key_fragment` be matched identically (`strict=True`) or matched by lower case comparison (`strict=False`)? Default: `False`

use_v1 *bool* : Chooses databroker API version between 'v1' or 'v2'. Default: `True` (meaning use the v1 API)

(new in apstools 1.5.1)

apstools.utils.list_runs.**listruns**(*cat=None*, *keys=None*, *missing=''*, *num=20*, *printing='smart'*, *reverse=True*, *since=None*, *sortby='time'*, *tablefmt='dataframe'*, *timefmt='%Y-%m-%d %H:%M:%S'*, *until=None*, *ids=None*, ***query*)

List runs from catalog.

This function provides a thin interface to the highly-reconfigurable `ListRuns()` class in this package.

PARAMETERS

cat *object* : Instance of databroker v1 or v2 catalog.

keys *str* or *[str]* or None: Include these additional keys from the start document. (default: `None` means `"scan_id time plan_name detectors"`)

missing *str*: Test to report when a value is not available. (default: `""`)

ids *[int]* or *[str]*: List of `uid` or `scan_id` value(s). Can mix different kinds in the same list. Also can specify offsets (e.g., `-1`). According to the rules for `databroker` catalogs, a string is a `uid` (partial representations allowed), an int is `scan_id` if positive or an offset if negative. (default: `None`)

num *int* : Make the table include the `num` most recent runs. (default: `20`)

printing *bool* or `"smart"`: If `True`, print the table to stdout. If `"smart"`, then act as shown below. (default: `True`)

session	action(s)
python session	print and return `None`
Ipython console	return `DataFrame` object
Jupyter notebook	return `DataFrame` object

reverse *bool* : If `True`, sort in descending order by `sortby`. (default: `True`)

since *str* : include runs that started on or after this ISO8601 time (default: `"1995-01-01"`)

sortby *str* : Sort columns by this key, found by exact match in either the `start` or `stop` document. (default: `"time"`)

tablefmt *str* : When returning an object, specify which type of object to return. (default: `"dataframe",`)

value	object
dataframe	`pandas.DataFrame`
table	`str(pyRestTable.Table)`

timefmt *str* : The `time` key (also includes keys `"start.time"` and `"stop.time"`) will be formatted by the `self.timefmt` value. See https://strftime.org/ for examples. The special `timefmt="raw"` is used to report time as the raw value (floating point time as used in python's `time.time()`). (default: `"%Y-%m-%d %H:%M:%S",`)

until *str* : include runs that started before this ISO8601 time (default: `2100-12-31`)

****query** *dict* : Any additional keyword arguments will be passed to the databroker to refine the search for matching runs using the `mongoquery` package.

RETURNS

object: `None` or `str` or `pd.DataFrame()` object

EXAMPLE:

```
TODO
```

(new in release 1.5.0)

`apstools.utils.list_runs.`**`summarize_runs`**`(`*since=None*, *db=None*`)`

Report bluesky run metrics from the databroker.

- How many different plans?
- How many runs?
- How many times each run was used?

- How frequently? (TODO:)

PARAMETERS

since *str* : Report all runs since this ISO8601 date & time (default: `1995`)

db *object* : Instance of `databroker.Broker()` (default: `db` from the IPython shell)

Diagnostic Support for Memory

rss_mem()	return memory used by this process

apstools.utils.memory.**rss_mem**()
return memory used by this process

Miscellaneous Support

cleanupText(text)	convert text so it can be used as a dictionary key
connect_pvlist(pvlist[, wait, timeout, ...])	Given list of EPICS PV names, return dict of EpicsSignal objects.
dictionary_table(dictionary, **kwargs)	return a text table from `dictionary`
full_dotted_name(obj)	Return the full dotted name
itemizer(fmt, items)	Format a list of items.
listobjects([show_pv, printing, verbose, ...])	Show all the ophyd Signal and Device objects defined as globals.
pairwise(iterable)	break a list (or other iterable) into pairs
print_RE_md([dictionary, fmt, printing])	custom print the RunEngine metadata in a table
print_snapshot_list(db[, printing])	print (stdout) a list of all snapshots in the databroker
redefine_motor_position(motor, new_position)	Set EPICS motor record's user coordinate to `new_position`.
replay(headers[, callback, sort])	Replay the document stream from one (or more) scans (headers).
run_in_thread(func)	(decorator) run `func` in thread
safe_ophyd_name(text)	make text safe to be used as an ophyd object name
split_quoted_line(line)	splits a line into words some of which might be quoted
text_encode(source)	Encode `source` using the default codepoint.
to_unicode_or_bust(obj[, encoding])	from: http://farmdev.com/talks/unicode/ .
trim_string_for_EPICS(msg)	String must not exceed EPICS PV length.
unix(command[, raises])	Run a UNIX command, returns (stdout, stderr).

apstools.utils.misc.**cleanupText**(*text*)
convert text so it can be used as a dictionary key

Given some input text string, return a clean version remove troublesome characters, perhaps other cleanup as well. This is best done with regular expression pattern matching.

apstools.utils.misc.**connect_pvlist**(*pvlist*, *wait=True*, *timeout=2*, *poll_interval=0.1*)
Given list of EPICS PV names, return dict of EpicsSignal objects.

PARAMETERS

pvlist *[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)

apstools.utils.misc.**dictionary_table**(*dictionary*, ***kwargs*)
return a text table from `dictionary`

PARAMETERS

dictionary *dict* : Python dictionary

Note: Keyword arguments parameters are kept for compatibility with previous versions of apstools. They are ignored now.

RETURNS

table *object* or `None` : `pyRestTable.Table()` object (multiline text table) or `None` if dictionary has no contents

EXAMPLE:

```
In [8]: RE.md
Out[8]: {'login_id': 'jemian:wow.aps.anl.gov', 'beamline_id': 'developer',
 ↪'proposal_id': None, 'pid': 19072, 'scan_id': 10, 'version': {'bluesky': '1.5.2',
 ↪'ophyd': '1.3.3', 'apstools': '1.1.5', 'epics': '3.3.3'}}

In [9]: print(dictionary_table(RE.md))
==========␣
 ↪================================================================================
key         value
==========␣
 ↪================================================================================
beamline_id developer
login_id    jemian:wow.aps.anl.gov
pid         19072
proposal_id None
scan_id     10
version     {'bluesky': '1.5.2', 'ophyd': '1.3.3', 'apstools': '1.1.5', 'epics': '3.
 ↪3.3'}
==========␣
 ↪================================================================================
```

apstools.utils.misc.**full_dotted_name**(*obj*)
Return the full dotted name

The `.dotted_name` property does not include the name of the root object. This routine adds that.

see: https://github.com/bluesky/ophyd/pull/797

apstools.utils.misc.**itemizer**(*fmt*, *items*)
Format a list of items.

apstools.utils.misc.**listobjects**(*show_pv=True*, *printing=True*, *verbose=False*, *symbols=None*)
Show all the ophyd Signal and Device objects defined as globals.

PARAMETERS

show_pv *bool* : If True, also show relevant EPICS PV, if available. (default: True)

printing *bool* : If True, print table to stdout. (default: True)

verbose *bool* : If True, also show `str(obj`. (default: False)

symbols *dict* : If None, use global symbol table. If not None, use provided dictionary. (default: `globals()`)

RETURNS

object: Instance of `pyRestTable.Table()`

EXAMPLE:

```
In [1]: listobjects()
======== ================================ =============
name     ophyd structure                  EPICS PV
======== ================================ =============
adsimdet MySingleTriggerSimDetector       vm7SIM1:
m1       EpicsMotor                       vm7:m1
m2       EpicsMotor                       vm7:m2
m3       EpicsMotor                       vm7:m3
m4       EpicsMotor                       vm7:m4
m5       EpicsMotor                       vm7:m5
m6       EpicsMotor                       vm7:m6
m7       EpicsMotor                       vm7:m7
m8       EpicsMotor                       vm7:m8
noisy    EpicsSignalRO                    vm7:userCalc1
scaler   ScalerCH                        vm7:scaler1
shutter  SimulatedApsPssShutterWithStatus
======== ================================ =============

Out[1]: <pyRestTable.rest_table.Table at 0x7fa4398c7cf8>

In [2]:
```

(new in apstools release 1.1.8)

apstools.utils.misc.**pairwise**(*iterable*)

break a list (or other iterable) into pairs

```
s -> (s0, s1), (s2, s3), (s4, s5), ...

In [71]: for item in pairwise("a b c d e fg".split()):
    ...:     print(item)
    ...:
('a', 'b')
('c', 'd')
('e', 'fg')
```

apstools.utils.misc.**print_RE_md**(*dictionary=None*, *fmt='simple'*, *printing=True*)

custom print the RunEngine metadata in a table

PARAMETERS

dictionary *dict* : Python dictionary

EXAMPLE:

```
In [4]: print_RE_md()
RunEngine metadata dictionary:
======================= ================================
key                     value
```

(continues on next page)

(continued from previous page)

```
======================== ====================================
EPICS_CA_MAX_ARRAY_BYTES 1280000
EPICS_HOST_ARCH          linux-x86_64
beamline_id              APS USAXS 9-ID-C
login_id                 usaxs:usaxscontrol.xray.aps.anl.gov
pid                      67933
proposal_id              testing Bluesky installation
scan_id                  0
versions                 ======== =====
                         key      value
                         ======== =====
                         apstools 1.1.3
                         bluesky  1.5.2
                         epics    3.3.1
                         ophyd    1.3.3
                         ======== =====
======================== ====================================
```

apstools.utils.misc.**print_snapshot_list**(*db*, *printing=True*, ***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 apstools.utils import print_snapshot_list
    ...: from apstools.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)
```

(continues on next page)

(continued from previous page)

```
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',
    'databroker': '0.11.3',
    'apstools': '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
```

apstools.utils.misc.**redefine_motor_position**(*motor*, *new_position*)

Set EPICS motor record's user coordinate to `new_position`.

apstools.utils.misc.**replay**(*headers*, *callback=None*, *sort=True*)

Replay the document stream from one (or more) scans (headers).

PARAMETERS

headers *scan* or *[scan]* : Scan(s) to be replayed through callback. A *scan* is an instance of a Bluesky `databroker.Header`. see: https://nsls-ii.github.io/databroker/api.html?highlight=header#header-api

callback *scan* or *[scan]* : The Bluesky callback to handle the stream of documents from a scan. If `None`, then use the *bec* (BestEffortCallback) from the IPython shell. (default:`None`)

sort *bool* : Sort the headers chronologically if True. (default:`True`)

(new in apstools release 1.1.11)

apstools.utils.misc.**run_in_thread**(*func*)
(decorator) run func in thread

USAGE:

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

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

apstools.utils.misc.**safe_ophyd_name**(*text*)
make text safe to be used as an ophyd object name

Given some input text string, return a clean version. Remove troublesome characters, perhaps other cleanup as well. This is best done with regular expression pattern matching.

The "sanitized" name fits this regular expression:

```
[A-Za-z_][\w_]*
```

Also can be used for safe HDF5 and NeXus names.

apstools.utils.misc.**split_quoted_line**(*line*)
splits a line into words some of which might be quoted

TESTS:

```
FlyScan 0   0   0   blank
FlyScan 5   2   0   "empty container"
FlyScan 5   12   0   "even longer name"
SAXS 0 0 0 blank
SAXS 0 0 0 "blank"
```

RESULTS:

```
['FlyScan', '0', '0', '0', 'blank']
['FlyScan', '5', '2', '0', 'empty container']
['FlyScan', '5', '12', '0', 'even longer name']
['SAXS', '0', '0', '0', 'blank']
['SAXS', '0', '0', '0', 'blank']
```

apstools.utils.misc.**text_encode**(*source*)
Encode source using the default codepoint.

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

apstools.utils.misc.**trim_string_for_EPICS**(*msg*)
String must not exceed EPICS PV length.

apstools.utils.misc.**unix**(*command*, *raises=True*)
Run a UNIX command, returns (stdout, stderr).

PARAMETERS

command *str* : UNIX command to be executed

raises *bool* : If `True`, will raise exceptions as needed, default: `True`

OverrideParameters

Define parameters that can be overridden from a user configuration file.

EXAMPLE:

Create an `overrides` object in a new file `override_params.py`:

```
import apstools.utils
overrides = apstools.utils.OverrideParameters()
```

When code supports a parameter for which a user can provide a local override, the code should import the `overrides` object (from the `override_params` module), and then register the parameter name, such as this example:

```
from override_params import overrides
overrides.register("minimum_step")
```

Then later:

```
minstep = overrides.pick("minimum_step", 45e-6)
```

In the user's configuration file that will override the value of `45e-6` (such as can be loaded via `%run -i user.py`), import the *overrides* ` object (from the `override_params` module):

```
from override_params import overrides
```

and then override the attribute(s) as desired:

```
overrides.set("minimum_step", 1.0e-5)
```

With this override in place, the `minstep` value (from `pick()`) will be `1e-5`.

Get a pandas DataFrame object with all the overrides:

```
overrides.summary()
```

which returns this table:

```
    parameter    value
0  minimum_step  0.00001
```

OverrideParameters()	Define parameters that can be overridden from a user configuration file.

class apstools.utils.override_parameters.**OverrideParameters**

Define parameters that can be overridden from a user configuration file.

NOTE: This is a pure Python object, not using ophyd.

pick(parameter, default)	Return either the override parameter value if defined, or the default.
register(parameter_name)	Register a new parameter name to be supported by user overrides.
reset(parameter_name)	Remove an override value for a known parameter.
reset_all()	Remove override values for all known parameters.
set(parameter_name, value)	Define an override value for a known parameter.
summary()	Return a pandas DataFrame with all overrides.

(new in apstools 1.5.2)

pick(*parameter*, *default*)
Return either the override parameter value if defined, or the default.

register(*parameter_name*)
Register a new parameter name to be supported by user overrides.

reset(*parameter_name*)
Remove an override value for a known parameter. (sets it to undefined)

reset_all()
Remove override values for all known parameters. (sets all to undefined)

set(*parameter_name*, *value*)
Define an override value for a known parameter.

summary()
Return a pandas DataFrame with all overrides.

Parameter names that have no override value will be reported as `--undefined--`.

Plot Support

select_mpl_figure(x, y)	Get the MatPlotLib Figure window for y vs x.
select_live_plot(bec, signal)	Get the *first* live plot that matches `signal`.
trim_plot_lines(bec, n, x, y)	Find the plot with axes x and y and replot with at most the last *n* lines.
trim_plot_by_name([n, plots])	Find the plot(s) by name and replot with at most the last *n* lines.

apstools.utils.plot.**select_live_plot**(*bec*, *signal*)
Get the *first* live plot that matches `signal`.

PARAMETERS

bec *object*: instance of `bluesky.callbacks.best_effort.BestEffortCallback`

signal *object*: The Y axis object (an `ophyd.Signal`)

RETURNS

***object*:** Instance of `bluesky.callbacks.best_effort.LivePlotPlusPeaks()` or `None`

apstools.utils.plot.**select_mpl_figure**(*x*, *y*)
Get the MatPlotLib Figure window for y vs x.

PARAMETERS

x *object*: X axis object (an `ophyd.Signal`)

y ophyd object: X axis object (an `ophyd.Signal`)

RETURNS

object or None: Instance of `matplotlib.pyplot.Figure()`

apstools.utils.plot.**trim_plot_by_name**(*n=3*, *plots=None*)

Find the plot(s) by name and replot with at most the last *n* lines.

Note: this is not a bluesky plan. Call it as normal Python function.

It is recommended to call *trim_plot_by_name()* *before* the scan(s) that generate plots. Plots are generated from a RunEngine callback, executed *after* the scan completes.

PARAMETERS

n *int* : number of plots to keep

plots *str*, [*str*], or *None* : name(s) of plot windows to trim (default: all plot windows)

EXAMPLES:

```
trim_plot_by_name()   # default of n=3, apply to all plots
trim_plot_by_name(5)  # change from default of n=3
trim_plot_by_name(5, "noisy_det vs motor")  # just this plot
trim_plot_by_name(
    5,
    ["noisy_det vs motor", "det noisy_det vs motor"]]
)
```

EXAMPLE:

```
# use simulators from ophyd
from bluesky import plans as bp
from bluesky import plan_stubs as bps
from ophyd.sim import *

snooze = 0.25

def scan_set():
    trim_plot_by_name()
    yield from bp.scan([noisy_det], motor, -1, 1, 5)
    yield from bp.scan([noisy_det, det], motor, -2, 1, motor2, 3, 1, 6)
    yield from bps.sleep(snooze)

# repeat the_scans 15 times
uids = RE(bps.repeat(scan_set, 15))
```

(new in release 1.3.5)

apstools.utils.plot.**trim_plot_lines**(*bec*, *n*, *x*, *y*)

Find the plot with axes x and y and replot with at most the last *n* lines.

Note: *trim_plot_lines()* is not a bluesky plan. Call it as normal Python function.

EXAMPLE:

```
trim_plot_lines(bec, 1, m1, noisy)
```

PARAMETERS

bec *object* : instance of BestEffortCallback

n *int* : number of plots to keep

x *object* : instance of ophyd.Signal (or subclass), independent (x) axis

y *object* : instance of ophyd.Signal (or subclass), dependent (y) axis

(new in release 1.3.5)

Support for IPython profiles

getDefaultNamespace([attr])	get the IPython shell's namespace dictionary (or globals() if not found)
ipython_profile_name()	return the name of the current ipython profile or `None`
ipython_shell_namespace()	get the IPython shell's namespace dictionary (or empty if not found)

apstools.utils.profile_support.**getDefaultNamespace**(*attr='user_ns'*)
get the IPython shell's namespace dictionary (or globals() if not found)

apstools.utils.profile_support.**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'])
```

apstools.utils.profile_support.**ipython_shell_namespace**()
get the IPython shell's namespace dictionary (or empty if not found)

EPICS PV Registry

findbyname(oname[, force_rebuild, ns])	Find the ophyd (dotted name) object associated with the given ophyd name.
findbypv(pvname[, force_rebuild, ns])	Find all ophyd objects associated with the given EPICS PV.
PVRegistry([ns])	Cross-reference EPICS PVs with ophyd EpicsSignalBase objects.

class apstools.utils.pvregistry.**PVRegistry**(*ns=None*)
Cross-reference EPICS PVs with ophyd EpicsSignalBase objects.

ophyd_search(*oname*)
Search for ophyd object by ophyd name.

search(*pvname*)
Search for PV in both read & write modes.

search_by_mode(*pvname*, *mode='R'*)
Search for PV in specified mode.

apstools.utils.pvregistry.**findbyname**(*oname*, *force_rebuild=False*, *ns=None*)
Find the ophyd (dotted name) object associated with the given ophyd name.

PARAMETERS

oname *str* : ophyd name to search

force_rebuild *bool* : If `True`, rebuild the internal registry that maps ophyd names to ophyd objects.

ns *dict* or *None* : Namespace dictionary of Python objects.

RETURNS

str or None: Name of the ophyd object.

EXAMPLE:

```
In [45]: findbyname("adsimdet_cam_acquire")
Out[45]: 'adsimdet.cam.acquire'
```

(new in apstools 1.5.0)

apstools.utils.pvregistry.**findbypv**(*pvname*, *force_rebuild=False*, *ns=None*)
Find all ophyd objects associated with the given EPICS PV.

PARAMETERS

pvname *str* : EPICS PV name to search

force_rebuild *bool* : If `True`, rebuild the internal registry that maps EPICS PV names to ophyd objects.

ns *dict* or *None* : Namespace dictionary of Python objects.

RETURNS

dict or None: Dictionary of matching ophyd objects, keyed by how the PV is used by the ophyd signal. The keys are `read` and `write`.

EXAMPLE:

```
In [45]: findbypv("ad:cam1:Acquire")
Out[45]: {'read': [], 'write': ['adsimdet.cam.acquire']}

In [46]: findbypv("ad:cam1:Acquire_RBV")
Out[46]: {'read': ['adsimdet.cam.acquire'], 'write': []}
```

Searching databroker catalogs

db_query(db, query)	Searches the databroker v2 database.

apstools.utils.query.**db_query**(*db*, *query*)
Searches the databroker v2 database.

PARAMETERS

db *object* : Bluesky database, an instance of `databroker.catalog`.

query *dict* : Search parameters.

RETURNS

object **:** Bluesky database, an instance of `databroker.catalog` satisfying the `query` parameters.

See also:

`databroker.catalog.search()`

Common support of slits

SlitGeometry(width, height, x, y)	Slit size and center as a named tuple

class apstools.utils.slit_core.**SlitGeometry**(*width*, *height*, *x*, *y*)
Slit size and center as a named tuple

height
Alias for field number 1

width
Alias for field number 0

x
Alias for field number 2

y
Alias for field number 3

Spreadsheet Support

ExcelDatabaseFileBase([ignore_extra])	base class: read-only support for Excel files, treat them like databases
ExcelDatabaseFileGeneric(filename[, ...])	Generic (read-only) handling of Excel spreadsheet-as-database
ExcelReadError(*args, **kwargs)	Exception when reading Excel spreadsheet.

class apstools.utils.spreadsheet.**ExcelDatabaseFileBase**(*ignore_extra=True*)
base class: read-only support for Excel files, treat them like databases

Use this class when creating new, specific spreadsheet support.

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 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 unique key (row of the Excel file)'''
        key = entry["Name"]
        self.db[key] = entry
```

class apstools.utils.spreadsheet.**ExcelDatabaseFileGeneric**(*filename*, *labels_row=3*, *ignore_extra=True*)

Generic (read-only) handling of Excel spreadsheet-as-database

---

Note: This is the class to use when reading Excel spreadsheets.

In the spreadsheet, the first sheet should contain the table to be used. By default (see keyword parameter `labels_row`), the table should start in cell A4. The column labels are given in row 4. A blank column should appear to the right of the table (see keyword parameter `ignore_extra`). The column labels will describe the action and its parameters. Additional columns may be added for metadata or other purposes.

The rows below the column labels should contain actions and parameters for those actions, one action per row.

To make a comment, place a `#` in the action column. A comment should be ignored by the bluesky plan that reads this table. The table will end with a row of empty cells.

While it's a good idea to put the `action` column first, that is not necessary. It is not even necessary to name the column `action`. You can re-arrange the order of the columns and change their names **as long as** the column names match what text strings your Python code expects to find.

A future upgrade[1] will allow the table boundaries to be named by Excel when using Excel's `Format as Table`[2] feature. For now, leave a blank row and column at the bottom and right edges of the table.

PARAMETERS

filename *str* : name (absolute or relative) of Excel spreadsheet file

labels_row *int* : Row (zero-based numbering) of Excel file with column labels, default: `3` (Excel row 4)

ignore_extra *bool* : When `True`, ignore any cells outside of the table, default: `True`.

Note that when `True`, a row of cells *within* the table will be recognized as the end of the table, even if there are actions in following rows. To force an empty row, use a comment symbol `#` (actually, any non-empty content will work).

When `False`, cells with other information (in Sheet 1) will be made available, sometimes with unpredictable results.

EXAMPLE

See section *Example: the run_command_file() plan* for more examples.

(See also *example screen shot*.) Table (on Sheet 1) begins on row 4 in first column:

```
1  |  some text here, maybe a title
2  |  (could have content here)
3  |  (or even more content here)
4  |  action  | sx   | sy   | sample     | comments          |  | <-- leave empty␣
↪column
5  |  close   |      |                   | close the shutter |  |
6  |  image   | 0    | 0    | dark       | dark image        |  |
7  |  open    |      |      |            | open the shutter  |  |
8  |  image   | 0    | 0    | flat       | flat field image  |  |
9  |  image   | 5.1  | -3.2 | 4140 steel | heat 9172634      |  |
10 |  scan    | 5.1  | -3.2 | 4140 steel | heat 9172634      |  |
11 |  scan    | 0    | 0    | blank      |                   |  |
```

(continues on next page)

[1] https://github.com/BCDA-APS/apstools/issues/122

[2] Excel's `Format as Table`: https://support.office.com/en-us/article/Format-an-Excel-table-6789619F-C889-495C-99C2-2F971C0E2370

(continued from previous page)

```
12 |
13 |  ^^^ leave empty row ^^^
14 | (could have content here)
```

Example python code to read this spreadsheet:

```
from apstools.utils import ExcelDatabaseFileGeneric, cleanupText

def myExcelPlan(xl_file, md={}):
    excel_file = os.path.abspath(xl_file)
    xl = ExcelDatabaseFileGeneric(excel_file)
    for i, row in xl.db.values():
        # prepare the metadata
        _md = {cleanupText(k): v for k, v in row.items()}
        _md["xl_file"] = xl_file
        _md["excel_row_number"] = i+1
        _md.update(md) # overlay with user-supplied metadata

        # determine what action to take
        action = row["action"].lower()
        if action == "open":
            yield from bps.mv(shutter, "open")
        elif action == "close":
            yield from bps.mv(shutter, "close")
        elif action == "image":
            # your code to take an image, given **row as parameters
            yield from my_image(**row, md=_md)
        elif action == "scan":
            # your code to make a scan, given **row as parameters
            yield from my_scan(**row, md=_md)
        else:
            print(f"no handling for row {i+1}: action={action}")

# execute this plan through the RunEngine
RE(myExcelPlan("spreadsheet.xlsx", md=dict(purpose="apstools demo")))
```

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

class apstools.utils.spreadsheet.**ExcelReadError**(**args: Any*, ***kwargs: Any*)
Exception when reading Excel spreadsheet.

2.4.6 synApps Support: Records, Databases, ...

Ophyd-style support for EPICS synApps structures (records and databases).

EXAMPLES:

```
import apstools.synApps
calcs = apstools.synApps.userCalcsDevice("xxx:", name="calcs")
scans = apstools.synApps.SscanDevice("xxx:", name="scans")
scripts = apstools.synApps.userScriptsDevice("xxx:set1:", name="scripts")
```

(continues on next page)

(continued from previous page)

```
xxxstats = apstools.synApps.IocStatsDevice("xxx:", name="xxxstats")

calc1 = calcs.calc1
apstools.synApps.swait_setup_random_number(calc1)

apstools.synApps.swait_setup_incrementer(calcs.calc2)

calc1.reset()
```

Categories

Support the default structures as provided by the synApps template XXX[1] IOC. Also support, as needed, for structures from EPICS base.

Records

AsynRecord(*args, **kwargs)	EPICS asyn record support in ophyd
BusyRecord(*args, **kwargs)	EPICS synApps busy record
CalcoutRecord(*args, **kwargs)	EPICS base calcout record support in ophyd
EpidRecord(*args, **kwargs)	EPICS synApps epid record support in ophyd
LuascriptRecord(*args, **kwargs)	EPICS synApps luascript record: used as `$(P):userScript$(N)`
ScalcoutRecord(*args, **kwargs)	EPICS SynApps calc scalcout record support in ophyd
SscanRecord(*args, **kwargs)	EPICS synApps sscan record: used as `$(P):scan(N)`
SseqRecord(*args, **kwargs)	EPICS synApps sseq record support in ophyd
SubRecord(*args, **kwargs)	EPICS base sub record support in ophyd
SwaitRecord(*args, **kwargs)	EPICS synApps swait record: used as `$(P):userCalc$(N)`
TransformRecord(*args, **kwargs)	EPICS transform record support in ophyd

The ophyd-style Devices for these records rely on common structures:

EpicsRecordDeviceCommonAll(*args, **kwargs)	Many of the fields common to all EPICS records.
EpicsRecordInputFields(*args, **kwargs)	Some fields common to EPICS input records.
EpicsRecordOutputFields(*args, **kwargs)	Some fields common to EPICS output records.
EpicsRecordFloatFields(*args, **kwargs)	Some fields common to EPICS records supporting floating point values.

[1] synApps XXX: https://github.com/epics-modules/xxx

Databases

EditStringSequence(*args, **kwargs)	EPICS synApps sseq support to quickly re-arrange steps.
Optics2Slit1D(*args, **kwargs)	EPICS synApps optics 2slit.db 1D support: xn, xp, size, center, sync
Optics2Slit2D_HV(*args, **kwargs)	EPICS synApps optics 2slit.db 2D support: h.xn, h.xp, v.xn, v.xp
Optics2Slit2D_InbOutBotTop(*args, **kwargs)	EPICS synApps optics 2slit.db 2D support: inb, out, bot, top
SaveData(*args, **kwargs)	EPICS synApps saveData support.
SscanDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of sscan records: `$(P):scan$(N)`
UserCalcN(*args, **kwargs)	Single instance of the userCalcN database.
UserCalcsDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of userCalcs: `$(P):userCalc$(N)`
UserCalcoutDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of user calcouts: `$(P):userCalcOut$(N)`
UserCalcoutN(*args, **kwargs)	Single instance of the userCalcoutN database.
UserScalcoutDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of user scalcouts: `$(P):userStringCalc$(N)`
UserScalcoutN(*args, **kwargs)	Single instance of the userStringCalcN database.
UserScriptsDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of user lua scripts: `$(P):userScript$(N)`
UserStringSequenceDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of userStringSeqs: `$(P):userStringSeq$(N)`
UserStringSequenceN(*args, **kwargs)	Single instance of the userStringSeqN database.
UserAverageN(*args, **kwargs)	EPICS synApps XXX IOC setup of user average: `$(P):userAve$(N)`
UserAverageDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of user averaging sub records: `$(P):userAve$(N)`
UserTransformN(*args, **kwargs)	Single instance of the userTranN database.
UserTransformsDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of userTransforms: `$(P):userTran$(N)`

EpicsSynAppsRecordEnableMixin(*args, **kwargs)	Supports `{PV}Enable` feature from user databases.
CalcoutRecordChannel(*args, **kwargs)	channel of a calcout record: A-L
IocStatsDevice(*args, **kwargs)	synApps IOC stats
LuascriptRecordNumberInput(*args, **kwargs)	number input of a synApps luascript record: A-J
LuascriptRecordStringInput(*args, **kwargs)	string input of a synApps luascript record: AA-JJ
ScalcoutRecordNumberChannel(*args, **kwargs)	Number channel of an scalcout record: A-L
ScalcoutRecordStringChannel(*args, **kwargs)	String channel of an scalcout record: AA-LL
SubRecordChannel(*args, **kwargs)	Number channel of a sub record: A-L
SwaitRecordChannel(*args, **kwargs)	EPICS synApps synApps swait record: single channel [A-L]

Other Support

These functions configure calcout or swait records for certain algorithms.

setup_gaussian_calcout(calcout, ref_signal)	setup calcout for noisy Gaussian
setup_incrementer_calcout(calcout[, scan, limit])	setup calcout record as an incrementer
setup_lorentzian_calcout(calcout, ref_signal)	setup calcout record for noisy Lorentzian
setup_gaussian_swait(swait, ref_signal[, ...])	setup swait for noisy Gaussian
setup_incrementer_swait(swait[, scan, limit])	setup swait record as an incrementer
setup_lorentzian_swait(swait, ref_signal[, ...])	setup swait record for noisy Lorentzian
setup_random_number_swait(swait, **kw)	setup swait record to generate random numbers

All Submodules

EPICS Record support: common structures

Some fields are common[1] to all[2] or some EPICS records. These are defined as mixin classes for the various ophyd-style devices.

Ophyd support for fields common to all EPICS records

Public Structures

EpicsRecordDeviceCommonAll(*args, **kwargs)	Many of the fields common to all EPICS records.
EpicsRecordInputFields(*args, **kwargs)	Some fields common to EPICS input records.
EpicsRecordOutputFields(*args, **kwargs)	Some fields common to EPICS output records.
EpicsRecordFloatFields(*args, **kwargs)	Some fields common to EPICS records supporting floating point values.
EpicsSynAppsRecordEnableMixin(*args, **kwargs)	Supports `{PV}Enable` feature from user databases.

see https://wiki-ext.aps.anl.gov/epics/index.php/RRM_3-14_dbCommon

see https://wiki-ext.aps.anl.gov/epics/index.php/RRM_3-14_Common

class apstools.synApps._common.**EpicsRecordDeviceCommonAll**(**args: Any*, ***kwargs: Any*)
Many of the fields common to all EPICS records.

Some fields are not included because they are not interesting to an EPICS client or are already provided in other support.

class apstools.synApps._common.**EpicsRecordFloatFields**(**args: Any*, ***kwargs: Any*)
Some fields common to EPICS records supporting floating point values.

class apstools.synApps._common.**EpicsRecordInputFields**(**args: Any*, ***kwargs: Any*)
Some fields common to EPICS input records.

class apstools.synApps._common.**EpicsRecordOutputFields**(**args: Any*, ***kwargs: Any*)
Some fields common to EPICS output records.

class apstools.synApps._common.**EpicsSynAppsRecordEnableMixin**(**args: Any*, ***kwargs: Any*)
Supports `{PV}Enable` feature from user databases.

[1] https://docs.epics-controls.org/en/latest/guides/EPICS_Intro.html#ioc-database,
[2] https://epics.anl.gov/base/R7-0/6-docs/RecordReference.html

reset()
set all fields to default values

synApps asyn record

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

Ophyd support for the EPICS asyn record

Public Structures

AsynRecord(*args, **kwargs)	EPICS asyn record support in ophyd

see https://github.com/epics-modules/asyn

class apstools.synApps.asyn.**AsynRecord**(**args: Any*, ***kwargs: Any*)
EPICS asyn record support in ophyd

See https://epics.anl.gov/modules/soft/asyn/R4-36/asynRecord.html

See https://github.com/epics-modules/asyn

See https://epics.anl.gov/modules/soft/asyn/

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)	EPICS synApps busy record

class apstools.synApps.busy.**BusyRecord**(**args: Any*, ***kwargs: Any*)
EPICS synApps busy record

EPICS base calcout record

The `calcout` record is part of EPICS base: https://wiki-ext.aps.anl.gov/epics/index.php/RRM_3-14_Calcout

Ophyd support for the EPICS calcout record

https://wiki-ext.aps.anl.gov/epics/index.php/RRM_3-14_Calcout

Public Structures

UserCalcoutDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of user calcouts: `$(P):userCalcOut$(N)`
UserCalcoutN(*args, **kwargs)	Single instance of the userCalcoutN database.
CalcoutRecord(*args, **kwargs)	EPICS base calcout record support in ophyd
CalcoutRecordChannel(*args, **kwargs)	channel of a calcout record: A-L
setup_gaussian_calcout(calcout, ref_signal)	setup calcout for noisy Gaussian
setup_incrementer_calcout(calcout[, scan, limit])	setup calcout record as an incrementer

continues on next page

Table 100 – continued from previous page

setup_lorentzian_calcout(calcout, ref_signal)	setup calcout record for noisy Lorentzian

class apstools.synApps.calcout.**CalcoutRecord**(**args: Any*, ***kwargs: Any*)
EPICS base calcout record support in ophyd

reset()	set all fields to default values

See https://wiki-ext.aps.anl.gov/epics/index.php/RRM_3-14_Calcout

reset()
set all fields to default values

class apstools.synApps.calcout.**CalcoutRecordChannel**(**args: Any*, ***kwargs: Any*)
channel of a calcout record: A-L

reset()	set all fields to default values

reset()
set all fields to default values

class apstools.synApps.calcout.**UserCalcoutDevice**(**args: Any*, ***kwargs: Any*)
EPICS synApps XXX IOC setup of user calcouts: `$(P):userCalcOut$(N)`

reset()	set all fields to default values

calcout1
alias of *apstools.synApps.calcout.UserCalcoutN*

calcout10
alias of *apstools.synApps.calcout.UserCalcoutN*

calcout2
alias of *apstools.synApps.calcout.UserCalcoutN*

calcout3
alias of *apstools.synApps.calcout.UserCalcoutN*

calcout4
alias of *apstools.synApps.calcout.UserCalcoutN*

calcout5
alias of *apstools.synApps.calcout.UserCalcoutN*

calcout6
alias of *apstools.synApps.calcout.UserCalcoutN*

calcout7
alias of *apstools.synApps.calcout.UserCalcoutN*

calcout8
alias of *apstools.synApps.calcout.UserCalcoutN*

calcout9
alias of *apstools.synApps.calcout.UserCalcoutN*

reset()
set all fields to default values

class apstools.synApps.calcout.**UserCalcoutN**(**args: Any*, ***kwargs: Any*)
Single instance of the userCalcoutN database.

apstools.synApps.calcout.**setup_gaussian_calcout**(*calcout*, *ref_signal*, *center=0*, *width=1*, *scale=1*, *noise=0.05*)
setup calcout for noisy Gaussian

calculation:

```
D*(0.95+E*RNDM)/exp(((A-B)/C)^2)
```

PARAMETERS

calcout *object* : instance of *CalcoutRecord*

ref_signal *object* : instance of `EpicsSignal` used as `A`

center *float* : EPICS record field `B`, default = 0

width *float* : EPICS record field `C`, default = 1

scale *float* : EPICS record field `D`, default = 1

noise *float* : EPICS record field `E`, default = 0.05

apstools.synApps.calcout.**setup_incrementer_calcout**(*calcout*, *scan=None*, *limit=100000*)
setup calcout record as an incrementer

PARAMETERS

calcout *object* : instance of *CalcoutRecord*

scan *text* or *int* or `None` : any of the EPICS record `.SCAN` values, or the index number of the value, set to default if `None`, default: `.1 second`

limit *int* or `None` : set the incrementer back to zero when this number is reached (or passed), default: 100000

apstools.synApps.calcout.**setup_lorentzian_calcout**(*calcout*, *ref_signal*, *center=0*, *width=1*, *scale=1*, *noise=0.05*)
setup calcout record for noisy Lorentzian

calculation:

```
D*(0.95+E*RNDM)/(1+((A-B)/C)^2)
```

PARAMETERS

calcout *object* : instance of *CalcoutRecord*

ref_signal *object* : instance of `EpicsSignal` used as `A`

center *float* : EPICS record field `B`, default = 0

width *float* : EPICS record field `C`, default = 1

scale *float* : EPICS record field `D`, default = 1

noise *float* : EPICS record field `E`, default = 0.05

EPICS synApps optics 2slit.db

db_2slit: synApps optics 2slit.db

There are two implementations, corresponding to differing and competing opinions of how the support should be implemented.

Coordinates of `Optics2Slit2D_InbOutBotTop` (viewing from detector towards source):

```
   top
inb   out
   bot
```

Coordinates of `Optics2Slit2D_HV` (viewing from detector towards source):

```
    v.xp
h.xn    h.xp
    v.xn
```

Each blade[1] (in the XIA slit controller) travels in a _cartesian_ coordinate system. Positive motion moves a blade **outwards** (towards the `p` suffix). Negative motion moves towards the `n` suffix. Size and center are computed by the underlying EPICS support.

hsize = out - inb vsize = top - bot

USAGE:

```
slit1 = Optics2Slit2D_HV("gp:Slit1", name="slit1")
slit1.geometry = 0.1, 0.1, 0, 0  # moves the slits
print(slit1.geometry)

slit2 = Optics2Slit_InbOutBotTop("gp:Slit2", name="slit2")
slit2.geometry = 0.1, 0.1, 0, 0  # moves the slits
print(slit2.geometry)
```

Public Structures

Optics2Slit1D(*args, **kwargs)	EPICS synApps optics 2slit.db 1D support: xn, xp, size, center, sync
Optics2Slit2D_HV(*args, **kwargs)	EPICS synApps optics 2slit.db 2D support: h.xn, h.xp, v.xn, v.xp
Optics2Slit2D_InbOutBotTop(*args, **kwargs)	EPICS synApps optics 2slit.db 2D support: inb, out, bot, top

see https://github.com/epics-modules/optics

new in release 1.6.0

class apstools.synApps.db_2slit.**Optics2Slit1D**(**args: Any*, ***kwargs: Any*)

EPICS synApps optics 2slit.db 1D support: xn, xp, size, center, sync

"sync" is used to tell the EPICS 2slit database to synchronize the virtual slit values with the actual motor positions.

center

alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

[1] Note that the blade names here may be different than the EPICS support. The difference is to make the names of the blades consistent with other slits with the Bluesky framework.

size
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

xn
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

xp
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

class apstools.synApps.db_2slit.**Optics2Slit2D_HV**(**args: Any*, ***kwargs: Any*)
EPICS synApps optics 2slit.db 2D support: h.xn, h.xp, v.xn, v.xp

property geometry
Return the slit 2D size and center as a namedtuple.

h
alias of *apstools.synApps.db_2slit.Optics2Slit1D*

v
alias of *apstools.synApps.db_2slit.Optics2Slit1D*

class apstools.synApps.db_2slit.**Optics2Slit2D_InbOutBotTop**(**args: Any*, ***kwargs: Any*)
EPICS synApps optics 2slit.db 2D support: inb, out, bot, top

bot
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

property geometry
Return the slit 2D size and center as a namedtuple.

hcenter
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

hsize
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

inb
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

out
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

top
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

vcenter
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

vsize
alias of *apstools.devices.positioner_soft_done.PVPositionerSoftDone*

synApps epid record

The `epid` record is part of the `std` module: https://epics.anl.gov/bcda/synApps/std/epidRecord.html

Ophyd support for the EPICS epid record

Public Structures

EpidRecord(*args, **kwargs)	EPICS synApps epid record support in ophyd

see https://epics.anl.gov/bcda/synApps/std/epidRecord.html

class apstools.synApps.epid.**EpidRecord**(**args: Any*, ***kwargs: Any*)
EPICS synApps epid record support in ophyd

See https://epics.anl.gov/bcda/synApps/std/epidRecord.html

synApps IOC statistics

The synApps iocStats support is in the synApps `iocStats` module: https://github.com/epics-modules/iocStats

Ophyd support for the iocStats support

Public Structures

IocStatsDevice(*args, **kwargs)	synApps IOC stats

class apstools.synApps.iocstats.**IocStatsDevice**(**args: Any*, ***kwargs: Any*)
synApps IOC stats

synApps luascript record

see the synApps `luascript` module suppport: https://epics-lua.readthedocs.io/en/latest/luascriptRecord.html

Ophyd support for the EPICS synApps luascript record

EXAMPLES:

```
import apstools.synApps
scripts = apstools.synApps.UserScriptsDevice("xxx:", name="scripts")
scripts.reset()
```

UserScriptsDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of user lua scripts: `$(P):userScript$(N)`
LuascriptRecord(*args, **kwargs)	EPICS synApps luascript record: used as `$(P):userScript$(N)`
LuascriptRecordNumberInput(*args, **kwargs)	number input of a synApps luascript record: A-J
LuascriptRecordStringInput(*args, **kwargs)	string input of a synApps luascript record: AA-JJ

see https://epics-lua.readthedocs.io/en/latest/luascriptRecord.html

class apstools.synApps.luascript.**LuascriptRecord**(**args: Any*, ***kwargs: Any*)
EPICS synApps luascript record: used as `$(P):userScript$(N)`

reset()	set all fields to default values

reset()
set all fields to default values

class apstools.synApps.luascript.**LuascriptRecordNumberInput**(**args: Any*, ***kwargs: Any*)
number input of a synApps luascript record: A-J

reset()
set all fields to default values

class apstools.synApps.luascript.**LuascriptRecordStringInput**(**args: Any*, ***kwargs: Any*)
string input of a synApps luascript record: AA-JJ

reset()
set all fields to default values

class apstools.synApps.luascript.**UserScriptsDevice**(**args: Any*, ***kwargs: Any*)
EPICS synApps XXX IOC setup of user lua scripts: `$(P):userScript$(N)`

reset()	set all fields to default values

reset()
set all fields to default values

script0
alias of *apstools.synApps.luascript.LuascriptRecord*

script1
alias of *apstools.synApps.luascript.LuascriptRecord*

script2
alias of *apstools.synApps.luascript.LuascriptRecord*

script3
alias of *apstools.synApps.luascript.LuascriptRecord*

script4
alias of *apstools.synApps.luascript.LuascriptRecord*

script5
alias of *apstools.synApps.luascript.LuascriptRecord*

script6
alias of *apstools.synApps.luascript.LuascriptRecord*

script7
alias of *apstools.synApps.luascript.LuascriptRecord*

script8
alias of *apstools.synApps.luascript.LuascriptRecord*

script9
alias of *apstools.synApps.luascript.LuascriptRecord*

synApps save data

The synApps SaveData support is in the synApps `sscan` module: https://github.com/epics-modules/sscan

Ophyd support for the EPICS synApps saveData support

see: https://epics.anl.gov/bcda/synApps/sscan/sscanRecord.html

EXAMPLE:

```
from apstools.synApps import SaveData
save_data = SaveData("xxx:saveData_", name="save_data")
```

Public Structures

SaveData(*args, **kwargs)	EPICS synApps saveData support.

class apstools.synApps.save_data.**SaveData**(**args: Any*, ***kwargs: Any*)

EPICS synApps saveData support.

usage:

```
from apstools.synApps import SaveData
save_data = SaveData("xxx:saveData_", name="save_data")
```

reset()	

EPICS synApps calc scalcout record

The `scalcout` record is part of EPICS synApps `calc`: http://htmlpreview.github.io/?https://github.com/epics-modules/calc/blob/R3-6-1/documentation/calcDocs.html

Ophyd support for the EPICS scalcout record

http://htmlpreview.github.io/?https://github.com/epics-modules/calc/blob/R3-6-1/documentation/calcDocs.html

Public Structures

UserScalcoutDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of user scalcouts: `$(P):userStringCalc$(N)`
UserScalcoutN(*args, **kwargs)	Single instance of the userStringCalcN database.
ScalcoutRecord(*args, **kwargs)	EPICS SynApps calc scalcout record support in ophyd
ScalcoutRecordNumberChannel(*args, **kwargs)	Number channel of an scalcout record: A-L
ScalcoutRecordStringChannel(*args, **kwargs)	String channel of an scalcout record: AA-LL

class apstools.synApps.scalcout.**ScalcoutRecord**(**args: Any*, ***kwargs: Any*)

EPICS SynApps calc scalcout record support in ophyd

reset()	set all fields to default values

See http://htmlpreview.github.io/?https://github.com/epics-modules/calc/blob/R3-6-1/documentation/calcDocs.html

reset()

set all fields to default values

class apstools.synApps.scalcout.**ScalcoutRecordNumberChannel**(**args: Any*, ***kwargs: Any*)

Number channel of an scalcout record: A-L

reset()	set all fields to default values

reset()

set all fields to default values

class apstools.synApps.scalcout.**ScalcoutRecordStringChannel**(**args: Any*, ***kwargs: Any*)

String channel of an scalcout record: AA-LL

reset()	set all fields to default values

reset()
set all fields to default values

class apstools.synApps.scalcout.**UserScalcoutDevice**(**args: Any*, ***kwargs: Any*)
EPICS synApps XXX IOC setup of user scalcouts: `$(P):userStringCalc$(N)`

reset()	set all fields to default values

reset()
set all fields to default values

scalcout1
alias of *apstools.synApps.scalcout.UserScalcoutN*

scalcout10
alias of *apstools.synApps.scalcout.UserScalcoutN*

scalcout2
alias of *apstools.synApps.scalcout.UserScalcoutN*

scalcout3
alias of *apstools.synApps.scalcout.UserScalcoutN*

scalcout4
alias of *apstools.synApps.scalcout.UserScalcoutN*

scalcout5
alias of *apstools.synApps.scalcout.UserScalcoutN*

scalcout6
alias of *apstools.synApps.scalcout.UserScalcoutN*

scalcout7
alias of *apstools.synApps.scalcout.UserScalcoutN*

scalcout8
alias of *apstools.synApps.scalcout.UserScalcoutN*

scalcout9
alias of *apstools.synApps.scalcout.UserScalcoutN*

class apstools.synApps.scalcout.**UserScalcoutN**(**args: Any*, ***kwargs: Any*)
Single instance of the userStringCalcN database.

synApps sscan record

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

Ophyd support for the EPICS synApps sscan record

see: https://epics.anl.gov/bcda/synApps/sscan/SscanRecord.html

EXAMPLE:

```
import apstools.synApps
scans = apstools.synApps.SscanDevice("xxx:", name="scans")
scans.select_channels()     # only the channels configured in EPICS
```

Public Structures

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

Private Structures

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

class apstools.synApps.sscan.**SscanDevice**(**args: Any*, ***kwargs: Any*)

EPICS synApps XXX IOC setup of sscan records: `$(P):scan$(N)`

reset()	set all fields to default values
select_channels()	Select only the scans that are configured in EPICS

reset()

set all fields to default values

scan1

alias of *apstools.synApps.sscan.SscanRecord*

scan2

alias of *apstools.synApps.sscan.SscanRecord*

scan3

alias of *apstools.synApps.sscan.SscanRecord*

scan4

alias of *apstools.synApps.sscan.SscanRecord*

scanH

alias of *apstools.synApps.sscan.SscanRecord*

select_channels()

Select only the scans that are configured in EPICS

class apstools.synApps.sscan.**SscanRecord**(**args: Any*, ***kwargs: Any*)

EPICS synApps sscan record: used as `$(P):scan(N)`

defined_in_EPICS	True if will be used in EPICS
reset()	set all fields to default values
select_channels()	Select channels that are configured in EPICS

property defined_in_EPICS

True if will be used in EPICS

reset()

set all fields to default values

select_channels()

Select channels that are configured in EPICS

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

class apstools.synApps.sscan.**sscanDetector**(**args: Any*, ***kwargs: Any*)
detector of an EPICS sscan record

defined_in_EPICS	True if defined in EPICS
reset()	set all fields to default values

property defined_in_EPICS
True if defined in EPICS

reset()
set all fields to default values

class apstools.synApps.sscan.**sscanPositioner**(**args: Any*, ***kwargs: Any*)
positioner of an EPICS sscan record

defined_in_EPICS	True if defined in EPICS
reset()	set all fields to default values

property defined_in_EPICS
True if defined in EPICS

reset()
set all fields to default values

class apstools.synApps.sscan.**sscanTrigger**(**args: Any*, ***kwargs: Any*)
detector trigger of an EPICS sscan record

reset()	set all fields to default values

property defined_in_EPICS
True if defined in EPICS

reset()
set all fields to default values

synApps sseq record

The `sseq` (String Sequence) record is part of the `calc` module:

see https://htmlpreview.github.io/?https://raw.githubusercontent.com/epics-modules/calc/R3-6-1/documentation/sseqRecord.html

Ophyd support for the EPICS sseq (string sequence) record

Public Structures

EditStringSequence(*args, **kwargs)	EPICS synApps sseq support to quickly re-arrange steps.
SseqRecord(*args, **kwargs)	EPICS synApps sseq record support in ophyd
UserStringSequenceDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of userStringSeqs: `$(P):userStringSeq$(N)`
UserStringSequenceN(*args, **kwargs)	Single instance of the userStringSeqN database.

class apstools.synApps.sseq.**EditStringSequence**(**args: Any*, ***kwargs: Any*)
EPICS synApps sseq support to quickly re-arrange steps.

See the `editSseq_more` GUI screen for assistance.

class apstools.synApps.sseq.**SseqRecord**(**args: Any*, ***kwargs: Any*)
EPICS synApps sseq record support in ophyd

abort()	.ABORT is a push button.
reset()	set all fields to default values

See https://htmlpreview.github.io/?https://raw.githubusercontent.com/epics-modules/calc/R3-6-1/documentation/sseqRecord.html

abort()
.ABORT is a push button. Send a 1 to the PV to "push" it.

Push this button without a timeout from the .put() method.

reset()
set all fields to default values

class apstools.synApps.sseq.**UserStringSequenceDevice**(**args: Any*, ***kwargs: Any*)
EPICS synApps XXX IOC setup of userStringSeqs: `$(P):userStringSeq$(N)`

Note: This will connect more than 1,000 EpicsSignal objects!

reset()	set all fields to default values

reset()
set all fields to default values

sseq1
alias of *apstools.synApps.sseq.UserStringSequenceN*

sseq10
alias of *apstools.synApps.sseq.UserStringSequenceN*

sseq2
alias of *apstools.synApps.sseq.UserStringSequenceN*

sseq3
alias of *apstools.synApps.sseq.UserStringSequenceN*

sseq4
alias of *apstools.synApps.sseq.UserStringSequenceN*

sseq5
alias of *apstools.synApps.sseq.UserStringSequenceN*

sseq6
alias of *apstools.synApps.sseq.UserStringSequenceN*

sseq7
alias of *apstools.synApps.sseq.UserStringSequenceN*

sseq8
alias of *apstools.synApps.sseq.UserStringSequenceN*

sseq9
alias of *apstools.synApps.sseq.UserStringSequenceN*

class apstools.synApps.sseq.**UserStringSequenceN**(**args: Any*, ***kwargs: Any*)
Single instance of the userStringSeqN database.

class apstools.synApps.sseq.**sseqRecordStep**(**args: Any*, ***kwargs: Any*)
EPICS synApps sseq single step of an sseq record.

Step of a synApps sseq record: 1..10 (note: for 10, the PVs use "A")

reset()	set all fields to default values

reset()
set all fields to default values

EPICS base sub record & synApps userAve records

The sub record is part of EPICS base: https://epics.anl.gov/base/R7-0/6-docs/subRecord.html

The user average databases (`$(P)userAve$(N)`) are from synApps.

Ophyd support for the EPICS sub record

https://epics.anl.gov/base/R7-0/6-docs/subRecord.html

Public Structures

UserAverageN(*args, **kwargs)	EPICS synApps XXX IOC setup of user average: `$(P):userAve$(N)`
UserAverageDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of user averaging sub records: `$(P):userAve$(N)`
SubRecord(*args, **kwargs)	EPICS base sub record support in ophyd
SubRecordChannel(*args, **kwargs)	Number channel of a sub record: A-L

class apstools.synApps.sub.**SubRecord**(**args: Any*, ***kwargs: Any*)
EPICS base sub record support in ophyd

reset()	set all fields to default values

See http://htmlpreview.github.io/?https://github.com/epics-modules/calc/blob/R3-6-1/documentation/calcDocs.html

reset()
set all fields to default values

class apstools.synApps.sub.**SubRecordChannel**(**args: Any*, ***kwargs: Any*)
Number channel of a sub record: A-L

reset()	set all fields to default values

reset()
set all fields to default values

class apstools.synApps.sub.**UserAverageDevice**(**args: Any*, ***kwargs: Any*)
EPICS synApps XXX IOC setup of user averaging sub records: `$(P):userAve$(N)`

reset()	set all fields to default values

average1
alias of *apstools.synApps.sub.UserAverageN*

average10
alias of *apstools.synApps.sub.UserAverageN*

average2
alias of *apstools.synApps.sub.UserAverageN*

average3
alias of *apstools.synApps.sub.UserAverageN*

average4
alias of *apstools.synApps.sub.UserAverageN*

average5
alias of *apstools.synApps.sub.UserAverageN*

average6
alias of *apstools.synApps.sub.UserAverageN*

average7
alias of *apstools.synApps.sub.UserAverageN*

average8
alias of *apstools.synApps.sub.UserAverageN*

average9
alias of *apstools.synApps.sub.UserAverageN*

reset()
set all fields to default values

class apstools.synApps.sub.**UserAverageN**(**args: Any*, ***kwargs: Any*)
EPICS synApps XXX IOC setup of user average: `$(P):userAve$(N)`

This database uses a sub record for most features plus additional records to support done, acquire, clear, and other features.

It *uses* a sub record and other, hence not exactly a *SubRecord()*.

reset()
set all fields to default values

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 apstools.synApps
calcs = apstools.synApps.UserCalcsDevice("xxx:", name="calcs")
```

(continues on next page)

(continued from previous page)

```
calc1 = calcs.calc1
apstools.synApps.setup_random_number_swait(calc1)

apstools.synApps.setup_incrementer_swait(calcs.calc2)

calc1.reset()
```

UserCalcN(*args, **kwargs)	Single instance of the userCalcN database.
UserCalcsDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of userCalcs: `$(P):userCalc$(N)`
SwaitRecord(*args, **kwargs)	EPICS synApps swait record: used as `$(P):userCalc$(N)`
SwaitRecordChannel(*args, **kwargs)	EPICS synApps synApps swait record: single channel [A-L]
setup_random_number_swait(swait, **kw)	setup swait record to generate random numbers
setup_gaussian_swait(swait, ref_signal[, ...])	setup swait for noisy Gaussian
setup_lorentzian_swait(swait, ref_signal[, ...])	setup swait record for noisy Lorentzian
setup_incrementer_swait(swait[, scan, limit])	setup swait record as an incrementer

see https://htmlpreview.github.io/?https://raw.githubusercontent.com/epics-modules/calc/R3-6-1/documentation/swaitRecord.html

class apstools.synApps.swait.**SwaitRecord**(**args: Any*, ***kwargs: Any*)
EPICS synApps swait record: used as `$(P):userCalc$(N)`

reset()	set all fields to default values

reset()
set all fields to default values

class apstools.synApps.swait.**SwaitRecordChannel**(**args: Any*, ***kwargs: Any*)
EPICS synApps synApps swait record: single channel [A-L]

reset()
set all fields to default values

class apstools.synApps.swait.**UserCalcN**(**args: Any*, ***kwargs: Any*)
Single instance of the userCalcN database.

class apstools.synApps.swait.**UserCalcsDevice**(**args: Any*, ***kwargs: Any*)
EPICS synApps XXX IOC setup of userCalcs: `$(P):userCalc$(N)`

reset()	set all fields to default values

calc1
alias of *apstools.synApps.swait.UserCalcN*

calc10
alias of *apstools.synApps.swait.UserCalcN*

calc2
alias of *apstools.synApps.swait.UserCalcN*

calc3
alias of *apstools.synApps.swait.UserCalcN*

calc4
alias of *apstools.synApps.swait.UserCalcN*

calc5
alias of *apstools.synApps.swait.UserCalcN*

calc6
alias of *apstools.synApps.swait.UserCalcN*

calc7
alias of *apstools.synApps.swait.UserCalcN*

calc8
alias of *apstools.synApps.swait.UserCalcN*

calc9
alias of *apstools.synApps.swait.UserCalcN*

reset()
set all fields to default values

apstools.synApps.swait.**setup_gaussian_swait**(*swait*, *ref_signal*, *center=0*, *width=1*, *scale=1*, *noise=0.05*)
setup swait for noisy Gaussian

calculation: $D*(0.95+E*RNDM)/exp(((A-B)/C)^2)$

PARAMETERS

swait *object* : instance of *SwaitRecord*

ref_signal *object* : instance of `EpicsSignal` used as A

center *float* : instance of `EpicsSignal` used as B, default = 0

width *float* : instance of `EpicsSignal` used as C, default = 1

scale *float* : instance of `EpicsSignal` used as D, default = 1

noise *float* : instance of `EpicsSignal` used as E, default = 0.05

apstools.synApps.swait.**setup_incrementer_swait**(*swait*, *scan=None*, *limit=100000*)
setup swait record as an incrementer

PARAMETERS

swait *object* : instance of *SwaitRecord*

scan *text* or *int* or `None` any of the EPICS record `.SCAN` values, or the index number of the value, set to default if `None`, default: `.1 second`

limit *int* or `None` set the incrementer back to zero when this number is reached (or passed), default: 100000

apstools.synApps.swait.**setup_lorentzian_swait**(*swait*, *ref_signal*, *center=0*, *width=1*, *scale=1*, *noise=0.05*)
setup swait record for noisy Lorentzian

calculation: $D*(0.95+E*RNDM)/(1+((A-B)/C)^2)$

PARAMETERS

swait *object* : instance of *SwaitRecord*

ref_signal *object* : instance of `EpicsSignal` used as A

center *float* : instance of `EpicsSignal` used as B, default = 0

width *float* : instance of `EpicsSignal` used as C, default = 1

scale *float* : instance of `EpicsSignal` used as D, default = 1

noise *float* : instance of `EpicsSignal` used as E, default = 0.05

`apstools.synApps.swait.`**`setup_random_number_swait`**(*swait*, ***kw*)
setup swait record to generate random numbers

synApps transform record

The `transform` record is part of the `calc` module:

see https://htmlpreview.github.io/?https://raw.githubusercontent.com/epics-modules/calc/R3-6-1/documentation/TransformRecord.html#Fields

Ophyd support for the EPICS transform record

Public Structures

UserTransformN(*args, **kwargs)	Single instance of the userTranN database.
UserTransformsDevice(*args, **kwargs)	EPICS synApps XXX IOC setup of userTransforms: `$(P):userTran$(N)`
TransformRecord(*args, **kwargs)	EPICS transform record support in ophyd

`class apstools.synApps.transform.`**`TransformRecord`**(**args: Any*, ***kwargs: Any*)
EPICS transform record support in ophyd

reset()	set all fields to default values

See https://htmlpreview.github.io/?https://raw.githubusercontent.com/epics-modules/calc/R3-6-1/documentation/TransformRecord.html#Fields

`reset`()
set all fields to default values

`class apstools.synApps.transform.`**`UserTransformN`**(**args: Any*, ***kwargs: Any*)
Single instance of the userTranN database.

`class apstools.synApps.transform.`**`UserTransformsDevice`**(**args: Any*, ***kwargs: Any*)
EPICS synApps XXX IOC setup of userTransforms: `$(P):userTran$(N)`

`reset`()
set all fields to default values

`transform1`
alias of *apstools.synApps.transform.UserTransformN*

`transform10`
alias of *apstools.synApps.transform.UserTransformN*

`transform2`
alias of *apstools.synApps.transform.UserTransformN*

`transform3`
alias of *apstools.synApps.transform.UserTransformN*

`transform4`
alias of *apstools.synApps.transform.UserTransformN*

transform5
alias of *apstools.synApps.transform.UserTransformN*

transform6
alias of *apstools.synApps.transform.UserTransformN*

transform7
alias of *apstools.synApps.transform.UserTransformN*

transform8
alias of *apstools.synApps.transform.UserTransformN*

transform9
alias of *apstools.synApps.transform.UserTransformN*

class apstools.synApps.transform.**transformRecordChannel**(**args: Any*, ***kwargs: Any*)
channel of a synApps transform record: A-P

reset()	set all fields to default values

reset()
set all fields to default values

2.5 Change History

The project milestones describe the future plans.

2.5.1 1.6.1

release expected by 2021-01-26

Fixes

- Move `enable` Component out from synApps Record devices.
- Renew the unit tests for PVPositionerSoftDoneWithStop.

2.5.2 1.6.0

released 2021-01-20

Breaking Changes

- Moved `apsbss` support to new `apsbss` package (install with either `pip` or `conda`). See https://bcda-aps.github.io/apsbss/ for details.
- Can use Python 3.7 - 3.9. Cannot use Python 3.10 yet due to upstream limitation from databroker and intake packages.
- Moved `command_list_as_table()` from *utils* into `plans/command_list`.
- Removed `BusyStatus` from *apstools.synApps.busy*

- `callbacks/`: `DocumentCollectorCallback`, `document_contents_callback`, and `SnapshotReport` moved into `callbacks/`.
- `devices/`: Reorganized all devices, including `synApps/`, into `devices/` subpackage.
- `devices/`: `SynPseudoVoigt()` moved from `signals/` to `devices/`.
- `plans/`: Reorganized `plans.py` and `_plans/` into `plans/` subpackage.
- `snapshot/`: Moved `snapshot` application and related files to a subdirectory.
- `utils/`: Reorganized `utils.py` and `_utils/` into `utils/` subpackage.

New Features and/or Enhancements

- Add support for Eurotherm 2216e temperature controller
- Add support for Lakeshore 336 temperature controller
- Add support for Lakeshore 340 temperature controller
- Add support for synApps calc `scalcout` record.
- Add support for synApps calc `sseq` record.
- Add support for EPICS base `sub` record.
- Add support for synApps calc `userAve` database.
- Add support for synApps calc `userStringSeq` database.
- Add support for synApps calc `userStringCalc` database.
- Add support for synApps optics `2slit` database.

Fixes

- Convert `None` to `"null"` when saving `PeakStats` to stream.

Maintenance

Now testing with Python versions 3.7 - 3.9. (Can't use with Py3.10 yet due to upstream requirements.)

Update notebooks:

- `demo_specfile_example`
- `demo_tuneaxis`

Remove notebooks:

- `demo_specfile_databroker`

Deprecations

- Applications
  - *apstools_plan_catalog* application and related support.
- Devices
  - `ApsCycleComputedRO`
  - `move_energy()` method in `KohzuSeqCtl_Monochromator` class
  - `ProcessController`
- Utilities
  - `device_read2table`
  - `json_export`
  - `json_import`
  - `listdevice_1_5_2`
  - `listruns_v1_4`
  - `object_explorer`

Contributors

- Gilberto Fabbris
- Jan Ilavsky
- Qingteng Zhang

2.5.3 1.5.4

release expected 2021-12-01

NOTE: The `apsbss` component will be moved out of `apstools` into its own package with the next release (1.6.0, ~Feb 2022) of `apstools`.

Notice

The Python version is limited to 3.7 due to *aps-dm-api* package. Expect this limitation to be relaxed, allowing any Python 3.7 and higher with the 1.6.0 release.

Fixes

- Added table of APS run cycle dates. Use that if *aps-dm-api* not available.
- Restricted python version to 3.7 due to upstream *aps_dm_api* package.
- Rename name *uid* to *token* to avoid LGTM security false alert.

Deprecations

This support was marked as deprecated in release 1.5.4:

- `apstools.devices.ApsCycleComputedRO`

2.5.4 1.5.3

released 2021-10-15

Notice

The `apstools.beamtime` module and related content (includes `apsbss`) will be moved to a new repository for release 1.6.0. This will remove the requirement that the APS data management tools (package *aps-dm*, which only works on the APS computing network) be included. With this change, users will be able to `conda install apstools -c aps-anl-tag` on computers outside of the APS computing network.

Breaking Changes

- `apstools.utils.listdevice` has a new API (old version renamed to `listdevice_1_5_2`)

New Features and/or Enhancements

- Kohzu monochromator `energy`, `wavelength`, and `theta` each are now a `PVPositioner` (subclass).
- Linkam temperature controller CI94
- Linkam temperature controller T96
- Stanford Research Systems 570 current preamplifier
- Stanford Research Systems PTC10 temperature controller
- XIA PF4 filter now supports multiple PF4 units.
- Generalize that amplifiers will have a `gain` Component attribute.
- Generalize that temperature controllers will have a `temperature` Component attribute that is a positioner (subclass of `ophyd.PVPositioner`).
- Enhanced positioners for EPICS Devices: * `apstools.devices.PVPositionerSoftDone` * `apstools.devices.PVPositionerSoftDoneWithStop`

Fixes

- Fixed bug in `devices.ApsCycleComputedRO` and `devices.ApsCycleDM` involving `datetime`.

Maintenance

- Moved all device support into individual modules under *apstools._devices* because *apstools.devices* module was getting too big. Will refactor all with release 1.6.0.
- Add unit tests for `devices.ApsCycle*` Devices.
- Add EPICS IOCs (ADSimDetector and synApps xxx) to continuous integration for use in unit testing.
- Unit tests now use *pytest* package.
- Suppress certain warnings during unit testing.

Deprecations

This support will be removed in release 1.6.0:

- `apstools.beamtime` module and related content (includes `apsbss`) will be moved to a new repository
- `apstools.devices.ProcessController`
- `apstools.utils.device_read2table`
- `apstools.utils.listdevice_1_5_2`
- `apstools.utils.object_explorer`

Contributors

- Fanny Rodolakis
- Gilberto Fabbris
- Jan Ilavsky
- Qingteng Zhang
- 4-ID-C Polar
- 8-ID-I XPCS
- 9-ID-C USAXS

2.5.5 1.5.2 (and previous)

See this table for release change histories, highlighted by version control reference (pull request or issue):

1.5.2 released 2021-09-29

- Drop Codacy (https://app.codacy.com/gh/BCDA-APS/apstools) as no longer needed.
- **#540** Add `apstools.utils.listplans()` function.
- **#534** Add `apstools.utils.OverrideParameters` class. Hoisted from APS USAXS instrument.
- **#537** Enhancements to `apstools.utils.listruns()`:
  - Add search by list of `scan_id` or `uid` values.
  - Optimize search speed.

- #534 Add `apstools.plans.documentation_run()` plan. Hoisted from APS USAXS instrument.
- #528 Add `kind=` kwarg to synApps Devices.
- #539 Break `devices` into submodule `_devices`.

1.5.1 released 2021-07-22

- #522 Deprecate *apstools.devices.ProcessController*. Suggest *ophyd.PVPositioner* instead.
- #521 Enhancement: new functions: getRunData(), getRunDataValue(), getStreamValues() & listRunKeys()
- #518 Bug fixed: TypeError from summary() of CalcoutRecord
- #517 Added support for python 3.9.
- #514 Refactor 'SIGNAL.value' to 'SIGNAL.get()'

1.5.0 released 2021-04-02

- #504 comment Dropped support for python 3.6.
- #495 Dropped diffractometer support code.
- #511 & #497 Add `utils.findbyname()` and `utils.findbypv()` functions.
- #506 `spec2ophyd` can now read SPEC config files from APS 17BM
- #504 Overhaul of listruns() using pandas. Previous code renamed to listruns_v1_4().
- #503 Unit tests with data now used msgpack-backed databroker.
- #495 remove *hklpy* requirement since all diffractometer support code will be moved to [*hklpy*](https://github.com/bluesky/hklpy) package.

1.4.1 released: 2021-01-23

- add Area Detector configuration examples: Pilatus & Perkin-Elmer, both writing image to HDF5 file
- #488 use first trigger_mode when priming AD plugin
- #487 ensure spec2ophyd code is packaged

1.4.0 released: 2021-01-15

- #483 Python code style must pass `flake8` test.
- #482 specwriter: Fix bug when plan_args structure includes a numpy ndarray.
- #474 `apstools.utils.listruns()` now defaults to the current catalog in use.

  New functions:

  - `apstools.utils.getDatabase()`
  - `apstools.utils.getDefaultDatabase()`
- #472 Respond to changes in upstream packages.
  - package requirements
  - auto-detection of command list format (Excel or text)

  - use *openpyxl*[1] instead of *xlrd*[2] and *pandas*[3] to read Microsoft Excel *.xlsx* spreadsheet files
- #470 Area Detector plugin preparation & detection.
  - **apstools.devices.AD_plugin_primed()** re-written completely
  - **apstools.devices.AD_prime_plugin()** replaced by `apstools.devices.AD_prime_plugin2()`
- #463 Remove deprecated features.
  - `apstools.suspenders.SuspendWhenChanged()`
  - `apstools.utils.plot_prune_fifo()`
  - `apstools.utils.show_ophyd_symbols()`
  - `apstools.synapps.asyn.AsynRecord.binary_output_maxlength()`
  - `apstools.devices.AD_warmed_up()`
- #451 Undulator and Kohzu monochromator functionalities
  - `apstools.devices.ApsUndulator()`

    Adds some `Signal` components (such as setting *kind* kwarg) that are helpful in moving the undulator

1.3.9 released 2020-11-30

- #459 `apsbss`: list ESAFs & proposals from other cycles
- #457 `apstools.utils.rss_mem()`: show memory used by this process

1.3.8 released: 2020-10-23

- #449 diffractometer wh() shows extra positioners
- #446 utils: device_read2table() renamed to listdevice()
- #445 synApps: add Device for iocStats
- #437 diffractometer add pa() report
- #426 diffractometer add simulated diffractometers
- #425 BUG fixed: listruns() when no stop document
- #423 BUG fixed: apsbss IOC starter script

1.3.7 released: 2020-09-18

- #422 additional AD support from APS USAXS
- #421 wait for undulator when start_button pushed
- #418 apsbss: only update APS run cycle name after current cycle ends

1.3.6 released 2020-09-04

- #416 apsbss: allow iso8601 time strings to have *option* for fractional seconds
- #415 Get APS cycle name from official source

1.3.5 released 2020-08-25

- #406 replace `plot_prune_fifo()` with `trim_plot()` and `trim_plot_by_name()`

[1] https://openpyxl.readthedocs.io
[2] https://xlrd.readthedocs.io
[3] https://pandas.pydata.org

- #405 add Y1 & Z2 read-only signal to Kohzu Monochromator device
- #403 deprecate `SuspendWhenChanged()`

1.3.4 released 2020-08-14

- #400 resolve warnings and example documentation inconsistency
- #399 parse iso8601 date for py36
- #398 DiffractometerMixin: add wh() method
- #396 provide spec2ophyd application
- #394 add utils.copy_filtered_catalog()
- #392 RTD make parameter lists clearer
- #390 improve formatting of parameter list in RTD
- #388 add utils.quantify_md_key_use()
- #385 spec2ophyd: make entry point

1.3.3 released 2020-07-22

- #384 apsbss: print, not log from commands
- #382 spec2ophyd analyses

1.3.2 released 2020-07-20

- #380 apsbss: fix object references

1.3.1 released 2020-07-18

- #378 apsbss_ioc.sh: add checkup (keep-alive feature for the IOC)
- #376 apsbss: example beam line-specific shell scripts
- #375 apsbss: add PVs for numbers of users
- #374 apsbss_ophyd: addDeviceDataAsStream() from USAXS
- #373 account for time zone when testing datetime-based file name
- #371 update & simplify the travis-ci setup
- #369 spec2ophyd: handle NONE in SPEC counters
- #368 spec2ophyd: config file as command-line argument
- #367 apsbss: move ophyd import from main
- #364 apsbss: add PVs for ioc_host and ioc_user
- #363 Handle when mailInFlag not provided
- #360 prefer logging to print

1.3.0 release expected by 2020-07-15

- add NeXus writer callback
- add `apsbss` : APS experiment metadata support
- #351 apsbss: put raw info into PV
- #350 apsbss: clarify meaning of reported dates
- #349 apsbss: add “next” subcommand

- #347 some apbss files not published
- #346 publish fails to push conda packages
- #344 listruns() uses databroker v2 API
- #343 review and update requirements
- #342 summarize runs in databroker by plan_name and frequency
- #341 tools to summarize activity
- #340 update copyright year
- #339 resolve Codacy code review issues
- #338 unit tests are leaving directories undeleted
- #337 Document new filewriter callbacks
- #336 add NeXus file writer from USAXS
- #335 update requirements
- #334 support APS proposal & ESAF systems to provide useful metadata
- #333 access APS proposal and ESAF information
- #332 listruns(): use databroker v2 API
- #329 add NeXus writer base class from USAXS

1.2.6 released *2020-06-26*

- #331 listruns succeeds even when number of existing runs is less than requested
- #330 BUG: listruns: less than 20 runs in catalog
- #328 epid: add final_value (.VAL field)
- #327 epid: remove clock_ticks (.CT field)
- #326 BUG: epid failed to connect to .CT field
- #325 BUG: epid final_value signal not found
- #324 BUG: epid controlled_value signal name

1.2.5 released *2020-06-05*

- #322 add py38 to travis config
- #320 multi-pass tune should use FWHM for next scan
- #318 AxisTunerMixin is now subclass of DeviceMixinBase
- #317 BUG: USAXS can't tune motors
- #316 BUG: Error in asyn object definition
- #315 BUG: AttributeError from db.hs

1.2.3 released *2020-05-07*

- #314 fix ImportError about SignalRO
- #313 update packaging requirements

1.2.2 released *2020-05-06*

- **DEPRECATION** #306 *apstools.plans.show_ophyd_symbols()* will be removed by 2020-07-01. Use *apstools.plans.listobjects()* instead.
- #311 adapt to databroker v1
- #310 enhance listruns() search capabilities
- #308 manage diffractometer constraints
- #307 add diffractometer emhancements
- #306 rename show_ophyd_objects() as listobjects()
- #305 add utils.safe_ophyd_name()
- #299 set_lim() does not set low limit

1.2.1 released *2020-02-18* - bug fix

- #297 fix import error

1.2.0 released *2020-02-18* - remove deprecated functions

- #293 remove run_blocker_in_plan()
- #292 remove list_recent_scans()
- #291 remove unix_cmd()
- #288 add object_explorer() (from APS 8-ID-I)

1.1.19 released *2020-02-15*

- #285 add EpicsMotorResolutionMixin
- #284 adjust ophyd.EpicsMotor when motor limits changed from other EPICS client
- #283 print_RE_md() now returns a pyRestTable.Table object

1.1.18 released *2020-02-09*

- PyPI would not accept the 1.1.17 version: *filename has already been used*
- see release notes for 1.1.17

1.1.17 released *2020-02-09* - hot fixes

- #277 replace .value with .get()
- #276 update ophyd metadata after motor set_lim()
- #274 APS user operations could be in mode 1 OR 2

1.1.16 released *2019-12-05*

- #269 bug: shutter does not move when expected
- #268 add *redefine_motor_position()* plan
- #267 remove *lineup()* plan for now
- #266 bug fix for #265
- #265 refactor of #264
- #264 Limit number of traces shown on a plot - use a FIFO
- #263 *device_read2table()* should print unless optioned False
- #262 add *lineup()* plan (from APS 8-ID-I XPCS)

1.1.15 released *2019-11-21* : bug fixes, adds asyn record support

- **#259** resolve AssertionError from setup_lorentzian_swait
- **#258** swait record does not units, some other fields
- **#255** plans: resolve indentation error
- **#254** add computed APS cycle as signal
- **#252** synApps: add asyn record support

1.1.14 released *2019-09-03* : bug fixes, more synApps support

- **#246** synApps: shorten name from synApps_ophyd
- **#245** swait & calcout: change from *EpicsMotor* to any *EpicsSignal*
- **#240** swait: refactor swait record & userCalc support
- **#239** transform: add support for transform record
- **#238** calcout: add support for calcout record & userCalcOuts
- **#237** epid: add support for epid record
- **#234** utils: replicate the *unix()* command
- **#230** signals: resolve TypeError

1.1.13 released *2019-08-15* : enhancements, bug fix, rename

- **#226** writer: unit tests for empty #O0 & P0 control lines
- **#224** rename: list_recent_scans –> listscans
- **#222** writer: add empty #O0 and #P0 lines
- **#220** ProcessController: bug fix - raised TypeError

1.1.12 released *2019-08-05* : bug fixes & updates

- **#219** `ProcessController`: bug fixes
- **#218** `replay()`: sort chronological by default
- **#216** `replay()`: fails when not list

1.1.11 released *2019-07-31* : updates & new utility

- **#214** new: `apstools.utils.APS_utils.replay()`
- **#213** `list_recent_scans` show `exit_status`
- **#212** `list_recent_scans` show reconstructed scan command

1.1.10 released *2019-07-30* : updates & bug fix

- **#211** `devices` calls to superclass `__init__()`
- **#209** `devices` call to superclass `__init__()`
- **#207** `show_ophyd_symbols` also shows labels
- **#206** new: `apstools.utils.APS_utils.list_recent_scans()`
- **#205** `show_ophyd_symbols` uses ipython shell's namespace
- **#202** add `labels` attribute to enable `wa` and `ct` magic commands

1.1.9 released *2019-07-28* : updates & bug fix

- **#203** *SpecWriterCallback*: *#N* is number of data columns
- **#199** *spec2ophyd* handle CNTPAR:read_misc_1

1.1.8 released *2019-07-25* : updates

- **#196** *spec2ophyd* handle MOTPAR:read_misc_1
- **#194** new `show_ophyd_symbols` shows table of global ophyd `Signal` and `Device` instances
- **#193** *spec2ophyd* ignore None items in SPEC config file
- **#192** *spec2ophyd* handles VM_EPICS_PV in SPEC config file
- **#191** *spec2ophyd* handles PSE_MAC_MOT in SPEC config file
- **#190** *spec2ophyd* handles MOTPAR in SPEC config file

1.1.7 released 2019-07-04

- **DEPRECATION** *apstools.plans.run_blocker_in_plan()* will be removed by 2019-12-31. Do not write blocking code in bluesky plans.
- Dropped python 3.5 from supported versions
- **#175** move *plans.run_in_thread()* to *utils.run_in_thread()*
- **#168** new *spec2ophyd* migrates SPEC config file to ophyd setup
- **#166** *device_read2table()*: format *device.read()* results in a pyRestTable.Table
- **#161** *addDeviceDataAsStream()*: add Device as named document stream event
- **#159** convert xlrd.XLRDError into apstools.utils.ExcelReadError
- **#158** new `run_command_file()` runs a command list from text file or Excel spreadsheet

1.1.6 released *2019-05-26*

- **#156** add ProcessController Device
- **#153** print dictionary contents as table
- **#151** EpicsMotor support for enable/disable
- **#148** more LGTM recommendations
- **#146** LGTM code review recommendations
- **#143** filewriter fails to raise IOError
- **#141** ValueError during tune()

1.1.5 released *2019-05-14*

- **#135** add refresh button to snapshot GUI

1.1.4 released *2019-05-14*

- **#140** *event-model* needs at least v1.8.0
- **#139** `ValueError` in `_scan()`

1.1.3 released *2019-05-10*

- adds packaging dependence on event-model
- **#137** adds *utils.json_export()* and *utils.json_import()*

1.1.1 released *2019-05-09*

- adds packaging dependence on spec2nexus
- **#136** get json document stream(s)
- **#134** add build on travis-ci with py3.7
- **#130** fix conda recipe and pip dependencies (thanks to Maksim Rakitin!)
- **#128** SpecWriterCallback.newfile() problem with scan_id = 0
- **#127** fixed: KeyError from SPEC filewriter
- **#126** add uid to metadata
- **#125** SPEC filewriter scan numbering when "new" data file exists
- **#124** fixed: utils.trim_string_for_EPICS() trimmed string too long
- **#100** fixed: SPEC file data columns in wrong places

1.1.0 released *2019.04.16*

- change release numbering to Semantic Versioning (remove all previous tags and releases)
- batch scans using Excel spreadsheets
- bluesky_snapshot_viewer and bluesky_snapshot
- conda package available

2.6 License

```
Copyright (c) 2017-2022, UChicago Argonne, LLC

All Rights Reserved

apstools (previously: APS_BlueSky_tools)

BCDA, Advanced Photon Source, Argonne National Laboratory


OPEN SOURCE LICENSE

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.  Software changes,
   modifications, or derivative works, should be noted with comments and
   the author and organization's name.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. Neither the names of UChicago Argonne, LLC or the Department of Energy
   nor the names of its contributors may be used to endorse or promote
   products derived from this software without specific prior written
```

(continues on next page)

(continued from previous page)

```
   permission.

4. The software and the end-user documentation included with the
   redistribution, if any, must include the following acknowledgment:

   "This product includes software produced by UChicago Argonne, LLC
   under Contract No. DE-AC02-06CH11357 with the Department of Energy."

*****************************************************************************

DISCLAIMER

THE SOFTWARE IS SUPPLIED "AS IS" WITHOUT WARRANTY OF ANY KIND.

Neither the United States GOVERNMENT, nor the United States Department
of Energy, NOR uchicago argonne, LLC, nor any of their employees, makes
any warranty, express or implied, or assumes any legal liability or
responsibility for the accuracy, completeness, or usefulness of any
information, data, apparatus, product, or process disclosed, or
represents that its use would not infringe privately owned rights.

*****************************************************************************
```

2.7 See Also

apstools home	https://BCDA-APS.github.io/apstools
apstools source	https://github.com/BCDA-APS/apstools
apsbss home	https://BCDA-APS.github.io/apsbss
Bluesky home	https://blueskyproject.io/
Bluesky source	https://github.com/bluesky

2.8 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

A

B

C

D

E

F

G

H

I

N

O

P

Q

R

S

T

U

V

W

X

Y