apstools Documentation

Release 2019.0321.1+0.gb9f3d19.dirty

Pete R. Jemian

Sep 18, 2020

CONTENTS:

Various Python tools for use with BlueSky at the APS

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

CHAPTER

ONE

PACKAGE INFORMATION

author Pete R. Jemian

email jemian@anl.gov

documentation https://apstools.readthedocs.io

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

1.1 Applications

There are two command-line applications provided by apstools:

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

1.2 bluesky_snapshot

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

1.2.1 Example - command line

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

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

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

positional arguments:
  EPICS_PV              EPICS PV name
```

(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}
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
========================================

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'}
```

(continues on next page)

(continued from previous page)

```
time: 1545352108.8262713
uid: 7e77708e-9169-45ab-b2b6-4e31534d980a

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

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

1.2.2 Source code documentation

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

USAGE:

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

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

positional arguments:
  EPICS_PV              EPICS PV name

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

class apstools.snapshot.**Capturing**
capture stdout output from a Python function call

https://stackoverflow.com/a/16571630/1046449

class apstools.snapshot.**SnapshotGui**(*config=None*)
Browse and display snapshots in a Tkinter GUI

USAGE (from command line):

```
bluesky_snapshot_viewer
```

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

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

EXAMPLES:

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

Note that these are equivalent:

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

apstools.snapshot.**snapshot_gui**(*config=None*)
run the snapshot viewer

1.3 Examples

1.3.1 Example: `plan_catalog()`

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

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

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

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

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

plan Name of the plan that initiated this scan.

args Arguments to the plan that initiated this scan.

This is run as a linux console command:

```
apstools_plan_catalog | tee out.txt
```

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

```
================== ========= == ======================== ============================================================
date/time          short_uid id plan                     args
================== ========= == ======================== ============================================================
2019-02-19 17:04:56 f1caf5aa  1  scan                     detectors=['scaler'], num=15, args=['m1', -5, 5], per_step=None
2019-02-19 17:09:23 adbfd046  1  scan                     detectors=['scaler'], num=15, args=['m1', -5, 5], per_step=None
2019-02-19 17:10:38 481a04b2  1  TuneAxis.multi_pass_tune
2019-02-19 17:10:45 0d45103a  2  TuneAxis.multi_pass_tune
2019-02-19 17:10:49 30d80cb1  3  TuneAxis.multi_pass_tune
2019-02-19 17:10:52 c354fe37  4  TuneAxis.tune
2019-02-19 17:11:20 225eef4b  1  scan                     detectors=['scaler'], num=15, args=['m1', -5, 5], per_step=None
```

1.3.2 Example: `specfile_example()`

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

1.3.3 Example: Create a SPEC file from databroker

We'll use a Jupyter notebook to demonstrate the how to get a scan from the databroker and write it to a spec data file. Follow here: https://github.com/BCDA-APS/apstools/blob/master/docs/source/resources/demo_specfile_databroker.ipynb

1.3.4 Example: `nscan()`

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

1.3.5 Example: `TuneAxis()`

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

1.3.6 Source Code Documentation

demonstrate BlueSky callbacks

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

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

apstools_plan_catalog command-line application

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

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

Example:

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

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

1.3.7 Downloads

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

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

1.4 Callbacks

Callbacks that might be useful at the APS using BlueSky

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

continues on next page

Table 2 – continued from previous page

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

FILE WRITER CALLBACK

see SpecWriterCallback()

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

Will reset when it receives a *start* document.

EXAMPLE:

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

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

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

1.5 Devices

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

APS GENERAL SUPPORT

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

AREA DETECTOR SUPPORT

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

DETECTOR / SCALER SUPPORT

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

MOTORS, POSITIONERS, AXES, . . .

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

synApps records

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

OTHER SUPPORT

DualPf4FilterBox(*args, **kwargs)	Dual Xia PF4 filter boxes using support from synApps (using Al, Ti foils)
EpicsDescriptionMixin(*args, **kwargs)	add a record's description field to a Device, such as EpicsMotor
KohzuSeqCtl_Monochromator(*args, **kwargs)	synApps Kohzu double-crystal monochromator sequence control program
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
DeviceMixinBase(*args, **kwargs)	Base class for apstools Device mixin classes

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

PARAMETERS

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

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

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

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

EXAMPLE:

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

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

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

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

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

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

apstools.devices.**logger = <Logger apstools.devices (INFO)>**
for convenience

apstools.devices.**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.)

1.6 File Writers

BlueSky callback that writes SPEC data files

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

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

EXAMPLE : use as BlueSky callback:

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

EXAMPLE : use as writer from Databroker:

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

EXAMPLE : use as writer from Databroker with customizations:

```
from apstools.filewriters import SpecWriterCallback

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

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

class apstools.filewriters.**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 and appends scan to the file when the *stop* document is received.

Parameters

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

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

RE : 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
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.filewriters.**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 (default: `event`)] Bluesky RunEngine document type. One of: `start descriptor event resource datum stop`

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

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
```

(continues on next page)

(continued from previous page)

```
#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
#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
```

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
```

(continues on next page)

(continued from previous page)

```
-1.5236 -1.5236842105263158 9.467074871063232 9 15501.419687691103
-1.5028000000000001 -1.5026315789473683 9.667330741882324 10 29570.38936784884
-1.4816 -1.4815789473684209 9.867793798446655 10 55562.3437459487
-1.4604000000000001 -1.4605263157894737 10.067811012268066 10 89519.64275090238
-1.4396 -1.4394736842105262 10.268356084823608 10 97008.97190269837
-1.4184 -1.418421052631579 10.470621824264526 10 65917.29757650592
-1.3972 -1.3973684210526316 10.669955730438232 11 36203.46726798266
-1.3764 -1.3763157894736842 10.870310306549072 11 18897.64061096024
-1.3552 -1.3552631578947367 11.070487976074219 11 10316.223844200193
-1.3344 -1.3342105263157895 11.271018743515015 11 6540.179615556269
-1.3132000000000001 -1.313157894736842 11.4724280834198 11 4643.555421314616
-1.292 -1.2921052631578946 11.673305034637451 12 3533.8582404216445
-1.2712 -1.2710526315789474 11.874176025390625 12 2809.1872596809008
-1.25 -1.25 12.074703216552734 12 2285.9226305883626
#C Fri Nov 17 11:59:08 2017.  num_events_primary = 20
#C Fri Nov 17 11:59:08 2017.  time = 2017-11-17 11:59:08.324011
#C Fri Nov 17 11:59:08 2017.  exit_status = success
```

1.7 Plans

Plans that might be useful at the APS when using BlueSky

nscan(detectors, *motor_sets[, num, ...])	Scan over `n` variables moved together, each in equally spaced steps.
run_blocker_in_plan(blocker, *args[, ...])	plan: run blocking function `blocker_(*args, **kwargs)` from a Bluesky plan
run_in_thread(func)	(decorator) run `func` in thread
snapshot(obj_list[, stream, md])	bluesky plan: record current values of list of ophyd signals
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

class apstools.plans.**TuneAxis**(*signals*, *axis*, *signal_name=None*)

tune an axis with a signal

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

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

Then the `motor` could be tuned individually:

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

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

Example:

```
tuner = TuneAxis([det], axis)
live_table = LiveTable(["axis", "det"])
```

(continues on next page)

(continued from previous page)

```
RE(tuner.multi_pass_tune(width=2, num=9), live_table)
RE(tuner.tune(width=0.05, num=9), live_table)
```

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

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

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

BlueSky plan for tuning this axis with this signal

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

PARAMETERS

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

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

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

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

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

md [dict, optional] metadata

peak_detected()

returns True if a peak was detected, otherwise False

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

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

BlueSky plan to execute one pass through the current scan range

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

PARAMETERS

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

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

md [dict, optional] metadata

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

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

PARAMETERS

detectors [list] list of 'readable' objects

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

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

start [float] starting position of motor

finish [float] ending position of motor

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

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

md [dict, optional] metadata

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

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

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

PARAMETERS

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

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

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

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

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

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

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

RE(my_sleep())
```

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

bluesky plan: record current values of list of ophyd signals

PARAMETERS

obj_list [list] list of ophyd Signal or EpicsSignal objects

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

md [dict] metadata

apstools.plans.**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={}*)

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_ophyd.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*] (default: `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.

device_settings_stream [str or *None*] (default: `"settings"`) 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.

poll_delay_s [float] (default: 0.001 seconds) 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.

phase_timeout_s [float] (default: 60 seconds) 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`.

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"))

apstools.plans.**tune_axes**(*axes*)

BlueSky plan to tune a list of axes in sequence

EXAMPLE

Sequentially, tune a list of preconfigured axes:

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

1.8 Signals

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

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

1.9 Suspenders

(bluesky) custom support for pausing a running plan

SuspendWhenChanged(*args, **kwargs)	Bluesky suspender

1.10 Utilities

Various utilities

cleanupText(text)	convert text so it can be used as a dictionary key
connect_pvlist(pvlist[, wait, timeout, ...])	given a list of EPICS PV names, return a dictionary of EpicsSignal objects
EmailNotifications([sender])	send email notifications when requested
ExcelDatabaseFileBase()	base class: read-only support for Excel files, treat them like databases
ExcelDatabaseFileGeneric(filename[, labels_row])	Generic (read-only) handling of Excel spreadsheet-as-database
ipython_profile_name()	return the name of the current ipython profile or *None*
pairwise(iterable)	break a list (or other iterable) into pairs
print_snapshot_list(db, **search_criteria)	print (stdout) a list of all snapshots in the databroker
text_encode(source)	encode `source` using the default codepoint
to_unicode_or_bust(obj[, encoding])	from: http://farmdev.com/talks/unicode/
trim_string_for_EPICS(msg)	string must not be too long for EPICS PV
unix_cmd(command_list)	run a UNIX command, returns (stdout, stderr)

class apstools.utils.**EmailNotifications**(*sender=None*)

send email notifications when requested

use default OS mail utility (so no credentials needed)

class apstools.utils.**ExcelDatabaseFileBase**

base class: read-only support for Excel files, treat them like databases

EXAMPLE

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

```
class ExhibitorsDB(ExcelDatabaseFileBase):
    '''
    content for Exhibitors, vendors, and Sponsors from the Excel file
    '''
```

(continues on next page)

(continued from previous page)

```
    EXCEL_FILE = os.path.join("resources", "exhibitors.xlsx")
    LABELS_ROW = 2

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

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

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

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

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

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

PARAMETERS

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

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

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

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

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

Example (add to default RunEngine metadata):

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

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

USAGE:

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

EXAMPLE:

```
In [16]: from 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)
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
```

(continues on next page)

(continued from previous page)

```
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.**text_encode**(*source*)
encode `source` using the default codepoint

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

apstools.utils.**trim_string_for_EPICS**(*msg*)
string must not be too long for EPICS PV

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

1.11 synApps busy record

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

Ophyd support for the EPICS busy record

Public Structures

`busyRecord`(*args, **kwargs)	

class apstools.synApps_ophyd.busy.**BusyStatus**(*value*)
An enumeration.

1.12 synApps sscan record

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

Ophyd support for the EPICS synApps sscan record

see: https://epics.anl.gov/bcda/synApps/sscan/sscanRecord.html

EXAMPLE

import apstools.synApps_ophyd scans = apstools.synApps_ophyd.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)

continues on next page

Table 20 – continued from previous page

sscanDevice(*args, **kwargs)	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

1.13 synApps swait record

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

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

Ophyd support for the EPICS synApps swait record

EXAMPLES:;

> import apstools.synApps_ophyd calcs = apstools.synApps_ophyd.userCalcsDevice("xxx:", name="calcs")
>
> calc1 = calcs.calc1 apstools.synApps_ophyd.swait_setup_random_number(calc1)
>
> apstools.synApps_ophyd.swait_setup_incrementer(calcs.calc2)
>
> calc1.reset()

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

apstools.synApps_ophyd.swait.**swait_setup_gaussian**(*swait*, *motor*, *center=0*, *width=1*, *scale=1*, *noise=0.05*)

setup swait for noisy Gaussian

apstools.synApps_ophyd.swait.**swait_setup_incrementer**(*swait*, *scan=None*, *limit=100000*)

setup swait record as an incrementer

apstools.synApps_ophyd.swait.**swait_setup_lorentzian**(*swait*, *motor*, *center=0*, *width=1*, *scale=1*, *noise=0.05*)

setup swait record for noisy Lorentzian

apstools.synApps_ophyd.swait.**swait_setup_random_number**(*swait*, ***kw*)

setup swait record to generate random numbers

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

N

P

R

S

T

U

W