APS_BlueSky_tools Documentation

Release 0.0.40+0.g525e06f.dirty

Pete R. Jemian

Dec 21, 2018

Contents:

1 Package Information **3**
- 1.1 Demo . . . 3
- 1.2 Callbacks . . . 4
- 1.3 Devices . . . 5
- 1.4 Examples . . . 5
- 1.5 File Writers . . . 6
- 1.6 Plans . . . 9
- 1.7 bluesky_snapshot . . . 13
- 1.8 sscan record . . . 16
- 1.9 Suspenders . . . 17
- 1.10 swait record . . . 17
- 1.11 Utilities . . . 18

2 Indices and tables **23**

Python Module Index **25**

Various Python tools for use with BlueSky at the APS

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

CHAPTER 1

Package Information

author Pete R. Jemian

email jemian@anl.gov

documentation https://APS_BlueSky_tools.readthedocs.io

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

1.1 Demo

demonstrate a BlueSky callback that writes SPEC data files

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

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

Example:

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

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

1.2 Callbacks

Callbacks that might be useful at the APS using BlueSky

document_contents_callback(key, doc)	prints document contents
DocumentCollectorCallback()	BlueSky callback to collect *all* documents from most-recent plan
SnapshotReport(*args, **kwargs)	show the data from a `APS_BlueSky_Tools.plans.snapshot()`

FILE WRITER CALLBACK

see `SpecWriterCallback()`

class `APS_BlueSky_tools.callbacks.`**DocumentCollectorCallback**

BlueSky callback to collect *all* documents from most-recent plan

Will reset when it receives a *start* document.

EXAMPLE:

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

receiver(*key*, *document*)

keep all documents from recent plan in memory

class `APS_BlueSky_tools.callbacks.`**SnapshotReport**(**args*, ***kwargs*)

show the data from a `APS_BlueSky_Tools.plans.snapshot()`

Find most recent snapshot between certain dates:

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

Use as callback to a snapshot plan:

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

descriptor(*doc*)

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

print_report(*header*)

simplify the job of writing our custom data table

method: play the entire document stream through this callback

`APS_BlueSky_tools.callbacks.`**document_contents_callback**(*key*, *doc*)

prints document contents

1.3 Devices

1.4 Examples

Examples that might be useful at the APS using BlueSky

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

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

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

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

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

PARAMETERS

name [str] name of detector signal

motor [*Mover*] The independent coordinate

motor_field [str] name of *Mover* field

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

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

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

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

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

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

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

EXAMPLE

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

EXAMPLE

```
import numpy as np
from APS_BlueSky_tools.examples 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())
```

(continues on next page)

(continued from previous page)

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

1.5 File Writers

BlueSky callback that writes SPEC data files

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

EXAMPLE : use as BlueSky callback:

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

EXAMPLE : use as writer from Databroker:

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

EXAMPLE : use as writer from Databroker with customizations:

```
from APS_BlueSky_tools.filewriters import SpecWriterCallback

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

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

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

collect data from BlueSky RunEngine documents to write as SPEC data

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

Parameters

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

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

User Interface methods

receiver(key, document)	BlueSky callback: receive all documents for handling
newfile([filename, reset_scan_id, RE])	prepare to use a new SPEC data file
usefile(filename)	read from existing SPEC data file
make_default_filename()	generate a file name to be used as default
clear()	reset all scan data defaults
prepare_scan_contents()	format the scan for a SPEC data file
write_scan()	write the most recent (completed) scan to the file

Internal methods

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

bulk_events(*doc*)
handle *bulk_events* documents

clear()
reset all scan data defaults

datum(*doc*)
handle *datum* documents

descriptor(*doc*)
handle *descriptor* documents

prepare for primary scan data, ignore any other data stream

event(*doc*)
handle *event* documents

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

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

but don't create it until we have data

prepare_scan_contents()
format the scan for a SPEC data file

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

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

resource(*doc*)
handle *resource* documents

start(*doc*)
handle *start* documents

stop(*doc*)
handle *stop* documents

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

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

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

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

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

Example output from `SpecWriterCallback()`:

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

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

(continues on next page)

(continued from previous page)

```
-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
```

1.6 Plans

Plans that might be useful at the APS when using BlueSky

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

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

Procedure Registry

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

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

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

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

EXAMPLE:

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

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

(continues on next page)

(continued from previous page)

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

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

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


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

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

dir
tuple of procedure names

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

remove(*procedure*)
remove procedure from registry

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

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

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

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

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

Then the `motor` could be tuned individually:

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

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

Example:

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

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

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

BlueSky plan for tuning this axis with this signal

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

PARAMETERS

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

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

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

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

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

md [dict, optional] metadata

peak_detected()

returns True if a peak was detected, otherwise False

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

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

BlueSky plan to execute one pass through the current scan range

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

PARAMETERS

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

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

md [dict, optional] metadata

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

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

PARAMETERS

detectors [list] list of 'readable' objects

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

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

start [float] starting position of motor

finish [float] ending position of motor

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

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

md [dict, optional] metadata

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

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

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 (using `time.sleep` as blocking function):

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

Example (within a plan, using `time.sleep` as blocking function):

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

RE(my_sleep())
```

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

(decorator) run `func` in thread

USAGE:

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

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

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

bluesky plan: record current values of list of ophyd signals

PARAMETERS

obj_list [list] list of ophyd Signal or EpicsSignal objects

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

md [dict] metadata

APS_BlueSky_tools.plans.**tune_axes**(*axes*)
BlueSky plan to tune a list of axes in sequence

1.7 bluesky_snapshot

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

1.7.1 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]
                        EPICS_PV [EPICS_PV ...]

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

positional arguments:
  EPICS_PV          EPICS PV name

optional arguments:
  -h, --help        show this help message and exit
  -b BROKER_CONFIG  YAML configuration for databroker
  -m METADATA       additional metadata, enclose in quotes, such as -m
                    "purpose=just tuned, situation=routine"
```

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 otz:IOC_CPU_LOAD otz:SYS_CPU_LOAD
otz:IOC_CPU_LOAD               0.100096
otz:SYS_CPU_LOAD               8.25789
```

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

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

========================================
snapshot: 2018-12-20 18:18:27.052009
========================================

hints: {}
iso8601: 2018-12-20 18:18:27.052009
plan_description: archive snapshot of ophyd Signals (usually EPICS PVs)
plan_name: snapshot
plan_type: generator
purpose: example
scan_id: 1
```

(continues on next page)

(continued from previous page)

```
software_versions: {'python': '3.6.2 |Continuum Analytics, Inc.| (default, Jul 20␣
↪2017, 13:51:32) \n[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]', 'PyEpics': '3.3.1',
↪'bluesky': '1.4.1', 'ophyd': '1.3.0', 'databroker': '0.11.3', 'APS_Bluesky_Tools':
↪'0.0.37'}
time: 1545351507.0527153
uid: ab270806-0697-4056-ba44-80f7b462bedc

========================== ====== ================ ==================
timestamp                  source name            value
========================== ====== ================ ==================
2018-12-20 18:18:17.109931 PV     otz:IOC_CPU_LOAD 0.10009559468607492
2018-12-20 18:18:17.109927 PV     otz:SYS_CPU_LOAD 10.935451151721377
========================== ====== ================ ==================

exit_status: success
num_events: {'primary': 1}
run_start: ab270806-0697-4056-ba44-80f7b462bedc
time: 1545351507.0696447
uid: a0b5b4ff-d9a7-47ce-ace7-1bba818da77b
```

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

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

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

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

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

exit_status: success
num_events: {'primary': 1}
run_start: d5e15ba3-0393-4df3-8217-1b72d82b5cf9
time: 1545351713.3957422
```

(continues on next page)

(continued from previous page)

```
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', 'APS_Bluesky_Tools':
↪'0.0.37'}
time: 1545352108.8262713
uid: 7e77708e-9169-45ab-b2b6-4e31534d980a

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

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

1.7.2 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]
                        EPICS_PV [EPICS_PV ...]

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

positional arguments:
  EPICS_PV          EPICS PV name

optional arguments:
  -h, --help        show this help message and exit
  -b BROKER_CONFIG  YAML configuration for databroker
  -m METADATA       additional metadata, enclose in quotes, such as -m
                    "purpose=just tuned, situation=routine"
```

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

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

EXAMPLES:

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

Note that these are equivalent:

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

1.8 sscan record

Ophyd support for the EPICS synApps sscan record

EXAMPLE

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

Public Structures

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

Private Structures

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

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

reset()
set all fields to default values

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

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

reset()
set all fields to default values

1.9 Suspenders

(bluesky) custom support for pausing a running plan

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

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

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

USAGE:

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

1.10 swait record

Ophyd support for the EPICS synApps swait record

EXAMPLES:;

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

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

APS_BlueSky_tools.synApps_ophyd.swait_setup_incrementer(calcs.calc2)

calc1.reset()

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

Continued on next page

Table 13 – continued from previous page

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

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

reset()
set all fields to default values

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

reset()
set all fields to default values

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

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

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

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

1.11 Utilities

Various utilities

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

Continued on next page

Table 14 – continued from previous page

unix_cmd(command_list)	run a UNIX command, returns (stdout, stderr)

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

use default OS mail utility (so no credentials needed)

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

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

EXAMPLE

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

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

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

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

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

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

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

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

PARAMETERS

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

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

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

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

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

Example (add to default RunEngine metadata):

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

APS_BlueSky_tools.utils.**print_snapshot_list**(*db*, ***search_criteria*)

print (stdout) a list of all snapshots in the databroker

USAGE:

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

EXAMPLE:

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

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

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

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

[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]', 'PyEpics': '3.3.1', 'bluesky': '1.4.1', 'ophyd': '1.3.0', 'databroker': '0.11.3', 'AI
time: 1545413970.063167 uid: 5049029d-075c-453c-96d2-55431273852b

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

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

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

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

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

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

APS_BlueSky_tools.callbacks, 4
APS_BlueSky_tools.demo, 3
APS_BlueSky_tools.examples, 5
APS_BlueSky_tools.filewriters, 6
APS_BlueSky_tools.plans, 9
APS_BlueSky_tools.snapshot, 15
APS_BlueSky_tools.suspenders, 17
APS_BlueSky_tools.synApps_ophyd.sscan, 16
APS_BlueSky_tools.synApps_ophyd.swait, 17
APS_BlueSky_tools.utils, 18

Index

A

add() (*APS_BlueSky_tools.plans.ProcedureRegistry method*), 10
APS_BlueSky_tools.callbacks (*module*), 4
APS_BlueSky_tools.demo (*module*), 3
APS_BlueSky_tools.examples (*module*), 5
APS_BlueSky_tools.filewriters (*module*), 6
APS_BlueSky_tools.plans (*module*), 9
APS_BlueSky_tools.snapshot (*module*), 15
APS_BlueSky_tools.suspenders (*module*), 17
APS_BlueSky_tools.synApps_ophyd.sscan (*module*), 16
APS_BlueSky_tools.synApps_ophyd.swait (*module*), 17
APS_BlueSky_tools.utils (*module*), 18

B

bulk_events() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7

C

clear() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7
connect_pvlist() (*in module APS_BlueSky_tools.utils*), 19

D

datum() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7
descriptor() (*APS_BlueSky_tools.callbacks.SnapshotReport method*), 4
descriptor() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7
dir (*APS_BlueSky_tools.plans.ProcedureRegistry attribute*), 10
document_contents_callback() (*in module APS_BlueSky_tools.callbacks*), 4
DocumentCollectorCallback (*class in APS_BlueSky_tools.callbacks*), 4

E

EmailNotifications (*class in APS_BlueSky_tools.utils*), 19
event() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7
ExcelDatabaseFileBase (*class in APS_BlueSky_tools.utils*), 19
ExcelDatabaseFileGeneric (*class in APS_BlueSky_tools.utils*), 19

G

get_args() (*in module APS_BlueSky_tools.snapshot*), 16

H

handleExcelRowEntry() (*APS_BlueSky_tools.utils.ExcelDatabaseFileGeneric method*), 19

I

ipython_profile_name() (*in module APS_BlueSky_tools.utils*), 19

M

make_default_filename() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7
multi_pass_tune() (*APS_BlueSky_tools.plans.TuneAxis method*), 11

N

newfile() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7
nscan() (*in module APS_BlueSky_tools.plans*), 12

P

peak_detected() (*APS_BlueSky_tools.plans.TuneAxis method*), 11

plan_catalog() (*in module APS_BlueSky_tools.demo*), 3
prepare_scan_contents() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7
print_report() (*APS_BlueSky_tools.callbacks.SnapshotReport method*), 4
print_snapshot_list() (*in module APS_BlueSky_tools.utils*), 19
ProcedureRegistry (*class in APS_BlueSky_tools.plans*), 9
put() (*APS_BlueSky_tools.plans.ProcedureRegistry method*), 10

R

receiver() (*APS_BlueSky_tools.callbacks.DocumentCollectorCallback method*), 4
receiver() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7
remove() (*APS_BlueSky_tools.plans.ProcedureRegistry method*), 10
reset() (*APS_BlueSky_tools.synApps_ophyd.sscan.sscanDevice method*), 17
reset() (*APS_BlueSky_tools.synApps_ophyd.sscan.sscanRecord method*), 16
reset() (*APS_BlueSky_tools.synApps_ophyd.swait.swaitRecord method*), 18
reset() (*APS_BlueSky_tools.synApps_ophyd.swait.userCalcsDevice method*), 18
resource() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7
run_blocker_in_plan() (*in module APS_BlueSky_tools.plans*), 12
run_in_thread() (*in module APS_BlueSky_tools.plans*), 12

S

send() (*APS_BlueSky_tools.utils.EmailNotifications method*), 19
set() (*APS_BlueSky_tools.plans.ProcedureRegistry method*), 10
set() (*APS_BlueSky_tools.synApps_ophyd.sscan.sscanRecord method*), 17
snapshot() (*in module APS_BlueSky_tools.plans*), 12
snapshot_cli() (*in module APS_BlueSky_tools.snapshot*), 16
SnapshotReport (*class in APS_BlueSky_tools.callbacks*), 4
specfile_example() (*in module APS_BlueSky_tools.demo*), 3
SpecWriterCallback (*class in APS_BlueSky_tools.filewriters*), 6
sscanDevice (*class in APS_BlueSky_tools.synApps_ophyd.sscan*), 17
sscanRecord (*class in APS_BlueSky_tools.synApps_ophyd.sscan*), 16
start() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7
stop() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 7
SuspendWhenChanged (*class in APS_BlueSky_tools.suspenders*), 17
swait_setup_gaussian() (*in module APS_BlueSky_tools.synApps_ophyd.swait*), 18
swait_setup_incrementer() (*in module APS_BlueSky_tools.synApps_ophyd.swait*), 18
swait_setup_lorentzian() (*in module APS_BlueSky_tools.synApps_ophyd.swait*), 18
swait_setup_random_number() (*in module APS_BlueSky_tools.synApps_ophyd.swait*), 18
swaitRecord (*class in APS_BlueSky_tools.synApps_ophyd.swait*), 18
SynPseudoVoigt (*class in APS_BlueSky_tools.examples*), 5

T

text_encode() (*in module APS_BlueSky_tools.utils*), 21
to_unicode_or_bust() (*in module APS_BlueSky_tools.utils*), 21
tune() (*APS_BlueSky_tools.plans.TuneAxis method*), 11
tune_axes() (*in module APS_BlueSky_tools.plans*), 13
TuneAxis (*class in APS_BlueSky_tools.plans*), 10

U

unix_cmd() (*in module APS_BlueSky_tools.utils*), 21
usefile() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 8
userCalcsDevice (*class in APS_BlueSky_tools.synApps_ophyd.swait*), 18

W

write_header() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 8
write_scan() (*APS_BlueSky_tools.filewriters.SpecWriterCallback method*), 8