Self-Describing Plans

Kyle Sunden
Blaise Thompson

University of Wisconsin–Madison

2022-04-01

Why does UW-Madison care?

- Want “first class” user interfaces for our Bluesky systems
  - typhos
  - webclient
- Mix of control systems
  - zero percent EPICS
- Way too small-scale to produce interfaces for each plan by hand
- Variety of plans
- Queueserver

Self-Describing Plans

Let's use annotations to make plans self-describing.

- automate GUI generation
- validation
- serialize
- help queueserver cast correctly

Prior Art

Queueserver already supports plan descrptions through a home-built annotation system.

```
"name": "annotated_count",
"properties": {"is_generator": true},
"parameters": [
    {"name": "detectors",
      "kind": {"name": "POSITIONAL_OR_KEYWORD", "value": 1}},
    {"name": "num",
      "kind": {"name": "POSITIONAL_OR_KEYWORD", "value": 1},
      "annotation": {"type": "int"},
      "default": "1"},
    {"name": "per_shot",
      "kind": {"name": "KEYWORD_ONLY", "value": 3},
      "default": "None"},
    {"name": "md",
      "kind": {"name": "KEYWORD_ONLY", "value": 3},
      "default": "None"}
],
"module": "__main__"
```

Prior Art

For those who need a review of parameter kinds enum (from inspect library)

0	POSITIONAL_ONLY
1	POSITIONAL_OR_KEYWORD
2	VAR_POSITIONAL
3	KEYWORD_ONLY
4	VAR_KEYWORD

Prior Art

From queueserver documentation

```
from ophyd.sim import det1, det2, det3
# Assume that the detectors 'det1', 'det2', 'det3' are in the list
#   of allowed devices for the user submitting the plan.

from bluesky_queueserver import parameter_annotation_decorator

@parameter_annotation_decorator({
    "parameters": {
        "detectors": {
            "annotation": "typing.List[DevicesType1]",
            "devices": {"DevicesType1": ["det1", "det2", "det3"]}
        }
    }
})
def plan_demo1f(detectors, npts):
    # The parameter 'detector_names'
    #   is expected to receive a list of detector names.
    <code implementing the plan>
```

Proposal

Use typing to fully annotate plans.
Simply annotate plans themselves as in PEP3107.

for count:

detectors	`typing.Sequence[bluesky.protocols.Readable]`
num	`int`
per_shot	`typing.Callable`
md	`typing.Dict[str, typing.Any]`

- use standard library features, less code to maintain
- static type checking
- easy to inspect for serialization
- doesn't support queueserver concept "limits", but extensible

Proposal

For those looking to automatically find all the plans in a namespace.
Annotate return type.

```
from typing import Generator
def count(detectors, num=1, delay=None, *, per_shot=None, md=None)
    -> Generator[Msg, None, None]:
```

Would be easy to check return annotation for one of:

```
Sequence[Msg]
Generator[Msg, Any, Any]
```

There are probably other valid annotations that I'm forgetting.
See bluesky/bluesky #1491

Proposal

Put it all together...

```
def count(detectors: typing.Sequence[bluesky.protocols.Readable],
          num: int = 1,
          delay: typing.Union[NoneType,
                              typing.Iterable[float],
                              float]
                                = None,
          *,
          per_shot: typing.Callable = None,
          md: typing.Dict[str, typing.Any] = None)
    -> Generator[Msg, None, None]:
```

Proposal

Problem: bluesky built-in plans make heavy use of variadic cycles.
Kyle and I cannot figure out how to hint these...
Relevant PEPs:

- PEP3107: Function Annotations
- PEP593: Flexible function and variable annotations
- PEP612: Parameter Specification Variables
- PEP613: Explicit Type Aliases
- PEP646: Variadic Generics
- PEP484: Type Hints

Proposal

Tuples for type checking, while remaining backwards compatible.

```
from bluesky import RunEngine
from bluesky import plans as bp
from ophyd.sim import det1, motor1, motor2
RE = RunEngine()
# still works
RE(bp.grid_scan([det1],
                motor1, -1.5, 1.5, 3,
                motor2, -0.1, 0.1, 5)
# now also valid
RE(bp.grid_scan([det1],
                (motor1, -1.5, 1.5, 3),
                (motor2, -0.1, 0.1, 5))
# passes mypy
RE(bp.grid_scan([det1],
                bp.GridScanAxis(motor1, -1.5, 1.5, 3),
                bp.GridScanAxis(motor2, -0.1, 0.1, 5))
```

Proposal

for count:

detectors	`typing.Sequence[bluesky.protocols.Readable]`
num	`int`
per_shot	`typing.Callable`
md	`typing.Dict[str, typing.Any]`

for grid_scan:

detectors	`typing.Sequence[bluesky.protocols.Readable]`
args	`GridScanAxis`
snake_axes	`typing.Optional[bool]`
per_step	`typing.Callable`
md	`typing.Dict[str, typing.Any]`

Proposal

Extend existing serialization system in Queueserver.
Migrate serialize function into bluesky library.

Cannot use existing schema standards, as far as we know...
objects are too complex.