Motivation

Proposal

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 for transport, RPC

State of the 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__"
```

Proposal

Use typing to fully annotate plans.
Simply annotate plans themselves with PEP3107-style planning.

for count:

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

- static type checking
- easy to inspect for serialization

Proposal

If annotated as yielding messages, can easily pick plans out of namespace.

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