Stetting up detectors using PSIDetectorBASE

base classes

class CustomDetectorMixing
init(self,*_args , parent :Device=None, **kwargs)
..... Implement detector specific logic here, such as
def on_stage()... Etc

class PSIDetectorBase(Device) [with Device importd from from ophyd]
Init(self,prefix,name,kind,parent,device_manager,**kwargs)
custon_prepare_cls = CustomDetectorMixing ← this brings name space of CustomDetectorMixing into PSIDetectorBase
Can be use foe example by
Custom_prepare_cls.on_stage()
To init detector we use init PSIDetectorBase, the 'mixing class is 'mixed in'.

derived Detector

class XXXSetup(CustomDetectorMixing) *Namespace* ← ***CustomDetectorMixing***
init(self,*_args , parent :Device=None, **kwargs) --- inherited
..... Implement detector specific logic here, such as
def on_stage()... Defined new funcions

class XXXDetector(PSIDetectorBase) *namespace* ← ***PSIDetectorBase*** ← ***Device***
Init(self,prefix,name,kind,parent,device_manager,**kwargs)
custon_prepare_cls = XXXSetup ← *this joins namespace space* ***of XXXSetup*** ← ***C ustomDetectorMixing***
custom_prepare_cls.on_stage() ← here he on_XXX- functions defined in XXX.Setup are called. All predefined in base classes
To init detector we use init PSIDetectorBase, the 'mixing class is 'mixed in'.

Connect to YAML file

deviceClass phoenix_bec.devices.{filename}.XXXDetector ← tell the device class
deviceConfig:
prefix : 'X07MB-ES1:' ← init parameter of device class

class OphydObject <w

def __init__(self, *, name=None-objectname}
attr_name=“”(attribute name of parent)=, parent =(objects parent name) labels=None, kind=None):
ophydobjects.py

Class BlueskyInterface

Defines basefunctionialitie, such as trigger, read, stage, nstage, etc.
ophyd/device.py

class Component(typing.Generic[K]):

"""A descriptor representing a device component (or signal)
Unrecognized keyword arguments will be passed directly to the component class initializer.
Parameters
def __init__(** self, **cls: Type[K], suffix: Optional[str] = None,
**, lazy: Optional[bool] = None, trigger_value: Optional[Any] = None, add_prefix: Optional[Sequence[str]] = None, doc: Optional[str] = None, kind: Union[str, Kind] = Kind.normal,*
***kwargs,*
---
cls : class ------ e.g. EpicsMotor
Class of signal to create. The required signature of `cls.__init__` is (if `suffix` is given)::
def __init__(self, pv_name, parent=None, **kwargs): ---- init of class cls , in the example this is EpicsMotor
or (if suffix is None) ::
def __init__(self, parent=None, **kwargs): --- init of class cls !!!!
suffix : str, optional ----------
The PV suffix, which gets appended onto ``parent.prefix`` to generate the final PV that the instance component will bind to.
Also see ``add_prefix``

class Device(BlueskyInterface, OphydObjec

def __init__(self, prefix="", ----- Device takes prefix
**, name, kind=None, read_attrs=None, configuration_attrs=Non*

class PositionerBase(OphydObject):

"""The positioner base class
Subclass from this to implement your own positioners.
"""
*def __init__(self, *, name=None, parent=None, settle_time=0.0, timeout=None, **kwargs):*
*super().__init__(name=name, parent=parent, **kwargs)*

x = Cpt(EpicsMotor, 'ScanX')

class Component(typing.Generic[K]):

def __init__(self, cls: Type[K], suffix: Optional[str] = None, …..)
Here we have cls = EpicsMotor
suffix = 'ScanX'
Class of signal to create. The required signature of
`cls.__init__` is
(if `suffix` is given)::
def __init__(self, pv_name, parent=None, **kwargs): ---- init of class cls , in the example this is EpicsMotor
or (if suffix is None) ::
def __init__(self, parent=None, **kwargs): --- init of class cls !!!!
suffix : str, optional ----------
The PV suffix, which gets appended onto ``parent.prefix`` to generate
the final PV that the instance component will bind to.
Also see ``add_prefix``

class EpicsMotor(Device, PositionerBase):
"""An EPICS motor record, wrapped in a :class:`Positioner`
Keyword arguments are passed through to the base class, Positioner
Parameters
----
prefix : str *----- prefix only as suffice are defined in class*
The record to use
read_attrs : sequence of attribute names
The signals to be read during data acquisition (i.e., in read() and describe() calls)
name : str, optional
The name of the device
parent : instance or None

```
def __init__(
self,
prefix="",
*,
name,
kind=None,
read_attrs=None,
configuration_attrs=None,
parent=None,
**kwargs
):
super().__init__(
prefix=prefix,
name=name,
kind=kind,
read_attrs=read_attrs,
configuration_attrs=configuration_attrs,
parent=parent,
**kwargs
```

Original PPTX file located G:\Huthwelker\000_CURRENT_PROJECTS\Space_for_Data_Exchange