ILD standard reconstruction with Gaudi

ILD Software/Analysis meeting

Thomas Madlener

Feb 28, 2024

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No 101004761.

Goals for migration to Gaudi

Why? Why now?

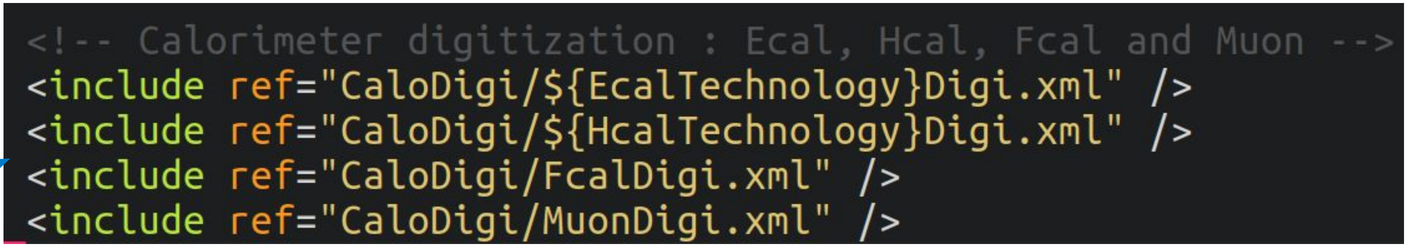
ILDConfig#137

- ILD reconstruction works in Key4hep
- Standard reconstruction workflow still based on Marlin and Marlin xml steering files
- Want to start migration towards Gaudi and more core Key4hep tools
  - **EDM4hep** as data format
- Key4hep comes with conversion tools to convert xml files to Gaudi python option files
  - Somewhat manual process still
  - Can convert everything once and then provide this centrally (less work for everyone 🎉)
- Decision at ILD workshop @CERN to start this migration
  - Also allows to more easily collaborate with CLD and others
- Major goal: **Keep existing functionality**
  - Everything that works now with Marlin should also work with Gaudi
  - Disclaimer: Some details will have to be defined after some usage experience

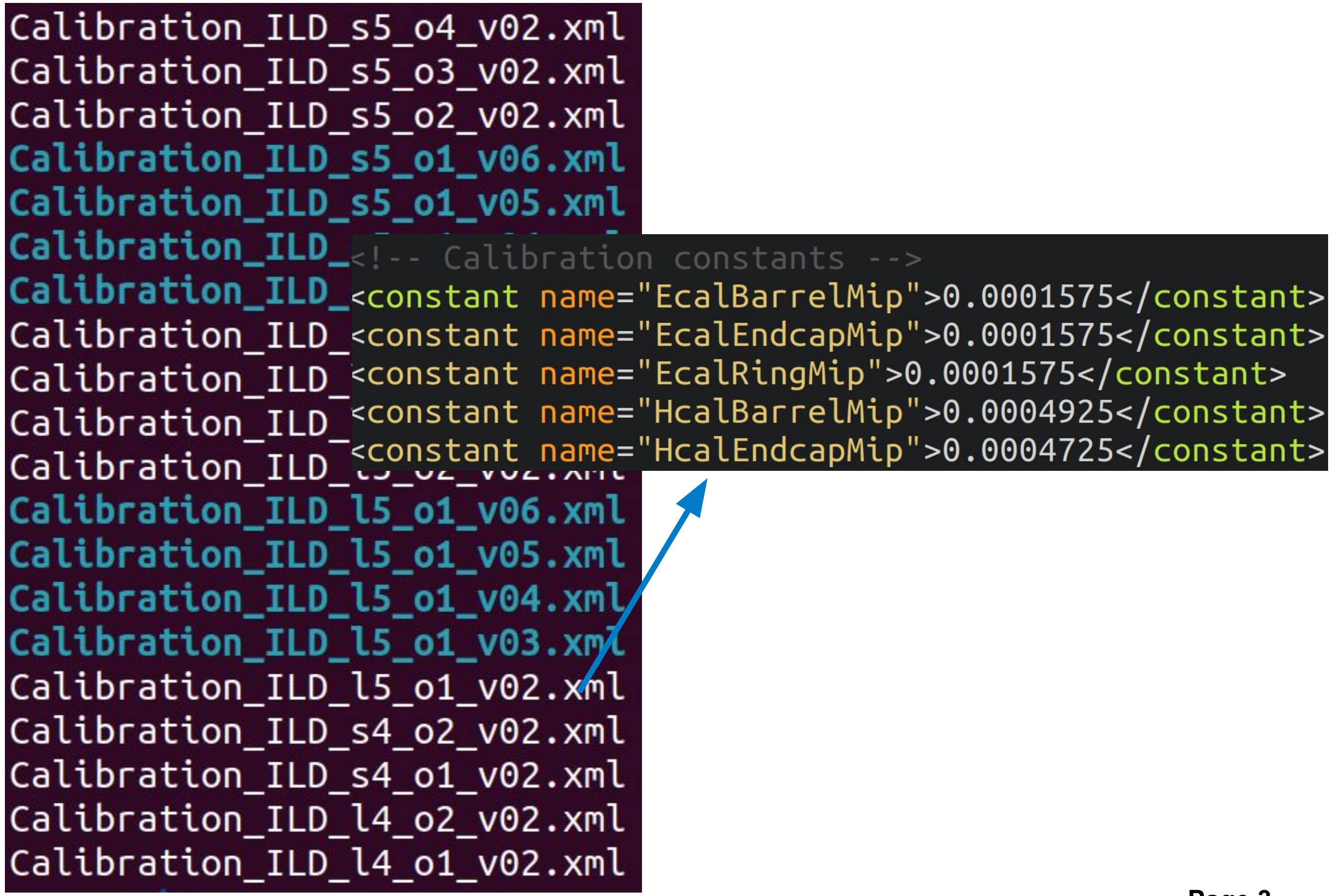
ILDConfig organization

Contents of StandardConfig/production

- Top level `MarlinStdReco.xml`
  - Incluces other steering and calibration files dynamically depending on values of `constants`
- Several subfolders with dedicated configuration for parts of the reconstruction
  - Tracking, Calorimetry, ParticleFlow, HighLevelReco
- Detector dependent calibration files
  - Define geometry and technology specific constants
- Center-of-mass energy dependent configuration files
  - Some minor cross-referencing with the former
- Some constants depend on the values of other constants



```
<!-- Calorimeter digitization : Ecal, Hcal, Fcal and Muon -->
<include ref="CaloDigi/${EcalTechnology}Digi.xml" />
<include ref="CaloDigi/${HcalTechnology}Digi.xml" />
<include ref="CaloDigi/FcalDigi.xml" />
<include ref="CaloDigi/MuonDigi.xml" />
```



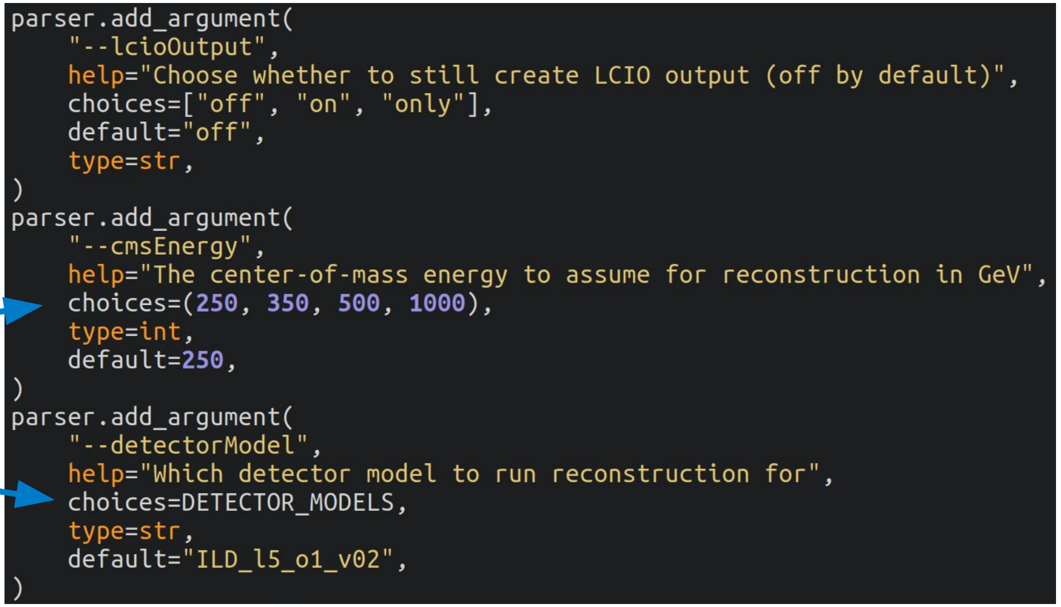
```
Calibration_ILD_s5_o4_v02.xml
Calibration_ILD_s5_o3_v02.xml
Calibration_ILD_s5_o2_v02.xml
Calibration_ILD_s5_o1_v06.xml
Calibration_ILD_s5_o1_v05.xml
Calibration_ILD_
Calibration_ILD_
Calibration_ILD_
Calibration_ILD_
Calibration_ILD_
Calibration_ILD_
Calibration_ILD_l5_o1_v06.xml
Calibration_ILD_l5_o1_v05.xml
Calibration_ILD_l5_o1_v04.xml
Calibration_ILD_l5_o1_v03.xml
Calibration_ILD_l5_o1_v02.xml
Calibration_ILD_s4_o2_v02.xml
Calibration_ILD_s4_o1_v02.xml
Calibration_ILD_l4_o2_v02.xml
Calibration_ILD_l4_o1_v02.xml
```

```
<!-- Calibration constants -->
<constant name="EcalBarrelMip">0.0001575</constant>
<constant name="EcalEndcapMip">0.0001575</constant>
<constant name="EcalRingMip">0.0001575</constant>
<constant name="HcalBarrelMip">0.0004925</constant>
<constant name="HcalEndcapMip">0.0004725</constant>
```

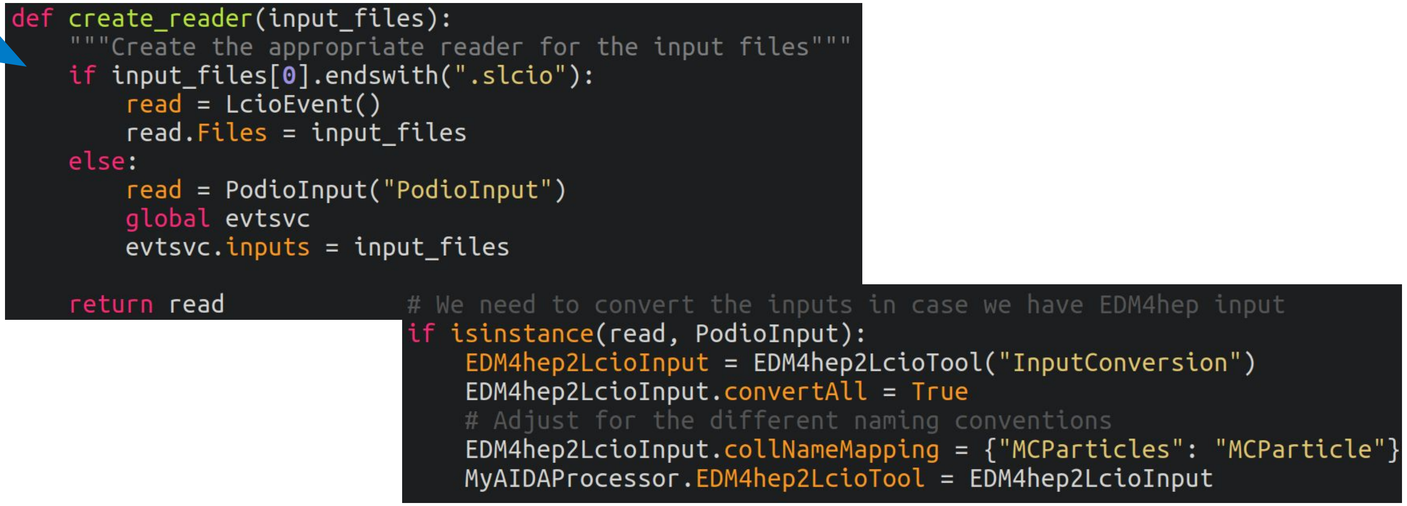
ILDReconstruction.py features

High level view

- Use `argparse` for argument parsing
  - Allows to catch configuration mistakes very early (e.g. non-existent calibration, …)
- Automatic input file format detection
  - Choose correct reader and if necessary inject a conversion directly
- EDM4hep output by default
  - Effectively equivalent to REC
  - LCIO output can be toggled via command line arguments
- Some new conventions on how to modularize configuration
  - Not yet fully finalized (i.e. voice your opinion now)



```
parser.add_argument(
    "--lcioOutput",
    help="Choose whether to still create LCIO output (off by default)",
    choices=["off", "on", "only"],
    default="off",
    type=str,
)
parser.add_argument(
    "--cmsEnergy",
    help="The center-of-mass energy to assume for reconstruction in GeV",
    choices=(250, 350, 500, 1000),
    type=int,
    default=250,
)
parser.add_argument(
    "--detectorModel",
    help="Which detector model to run reconstruction for",
    choices=DETECTOR_MODELS,
    type=str,
    default="ILD_l5_o1_v02",
)
```



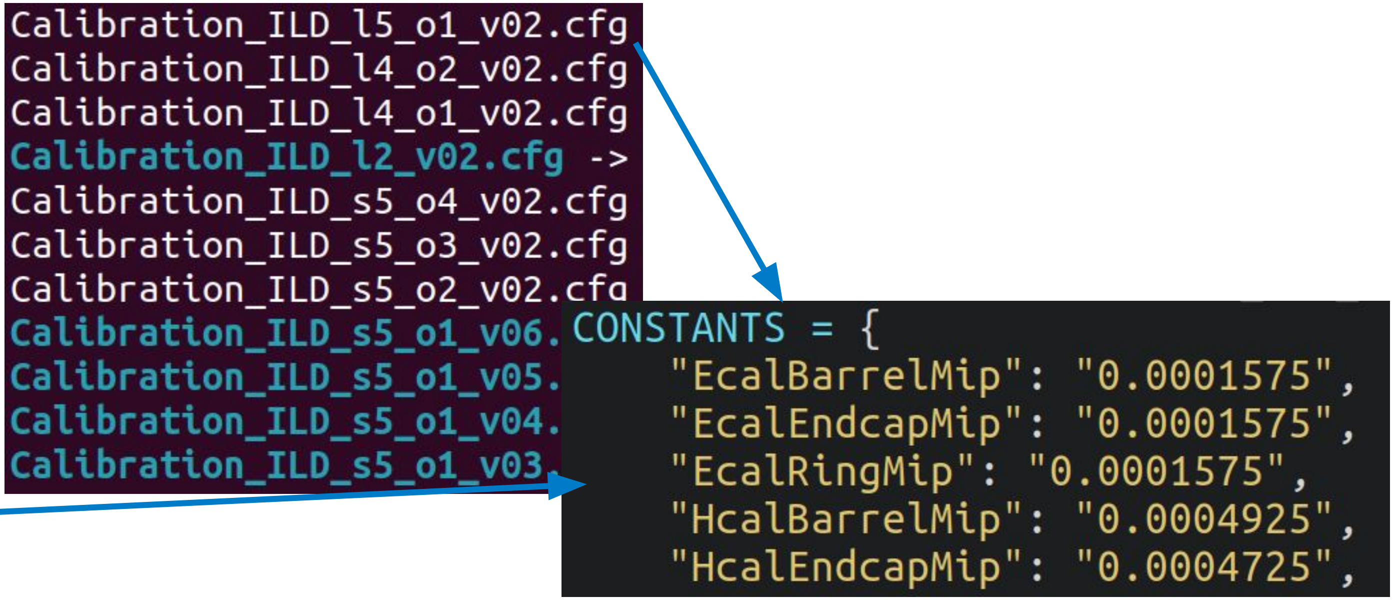
```
def create_reader(input_files):
    """Create the appropriate reader for the input files"""
    if input_files[0].endswith(".slcio"):
        read = LcioEvent()
        read.Files = input_files
    else:
        read = PodioInput("PodioInput")
        global evtsvc
        evtsvc.inputs = input_files

    return read
```

```
# We need to convert the inputs in case we have EDM4hep input
if isinstance(read, PodioInput):
    EDM4hep2LcioInput = EDM4hep2LcioTool("InputConversion")
    EDM4hep2LcioInput.convertAll = True
    # Adjust for the different naming conventions
    EDM4hep2LcioInput.collNameMapping = {"MCParticles": "MCParticle"}
    MyAIDAProcessor.EDM4hep2LcioTool = EDM4hep2LcioInput
```

Converted calibrations

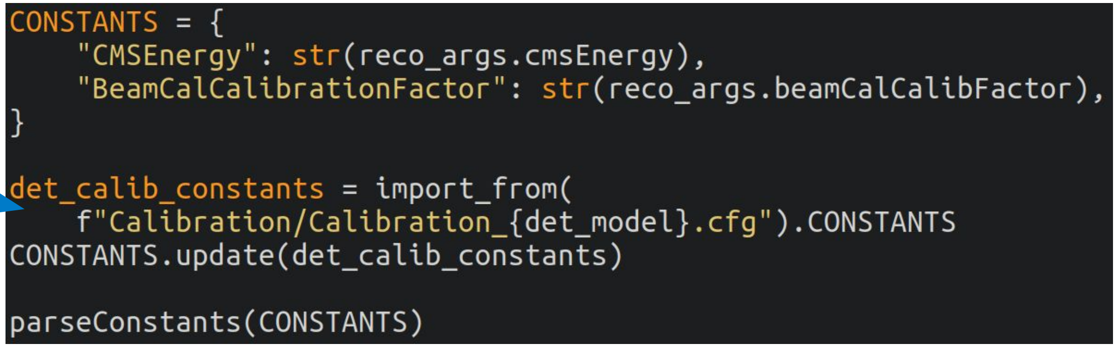
Same, same, but different

- Mirrored structure to existing XML calibration files
  - Simply converted all of them using converter script
  - Large python dictionary of constants
  - .cfg ending to signify that these are somewhat static configuration bits
- Dynamically imported into ILDReconstruction.py
  - Before nested constants are parsed
- Exactly the same mechanism for CMS energy dependent configuration
- Recursive resolution of nested constants in ILDReconstruction.py



```
Calibration_ILD_l5_o1_v02.cfg
Calibration_ILD_l4_o2_v02.cfg
Calibration_ILD_l4_o1_v02.cfg
Calibration_ILD_l2_v02.cfg ->
Calibration_ILD_s5_o4_v02.cfg
Calibration_ILD_s5_o3_v02.cfg
Calibration_ILD_s5_o2_v02.cfg
Calibration_ILD_s5_o1_v06.
Calibration_ILD_s5_o1_v05.
Calibration_ILD_s5_o1_v04.
Calibration_ILD_s5_o1_v03.
```

```
CONSTANTS = {
    "EcalBarrelMip": "0.0001575",
    "EcalEndcapMip": "0.0001575",
    "EcalRingMip": "0.0001575",
    "HcalBarrelMip": "0.0004925",
    "HcalEndcapMip": "0.0004725",
```

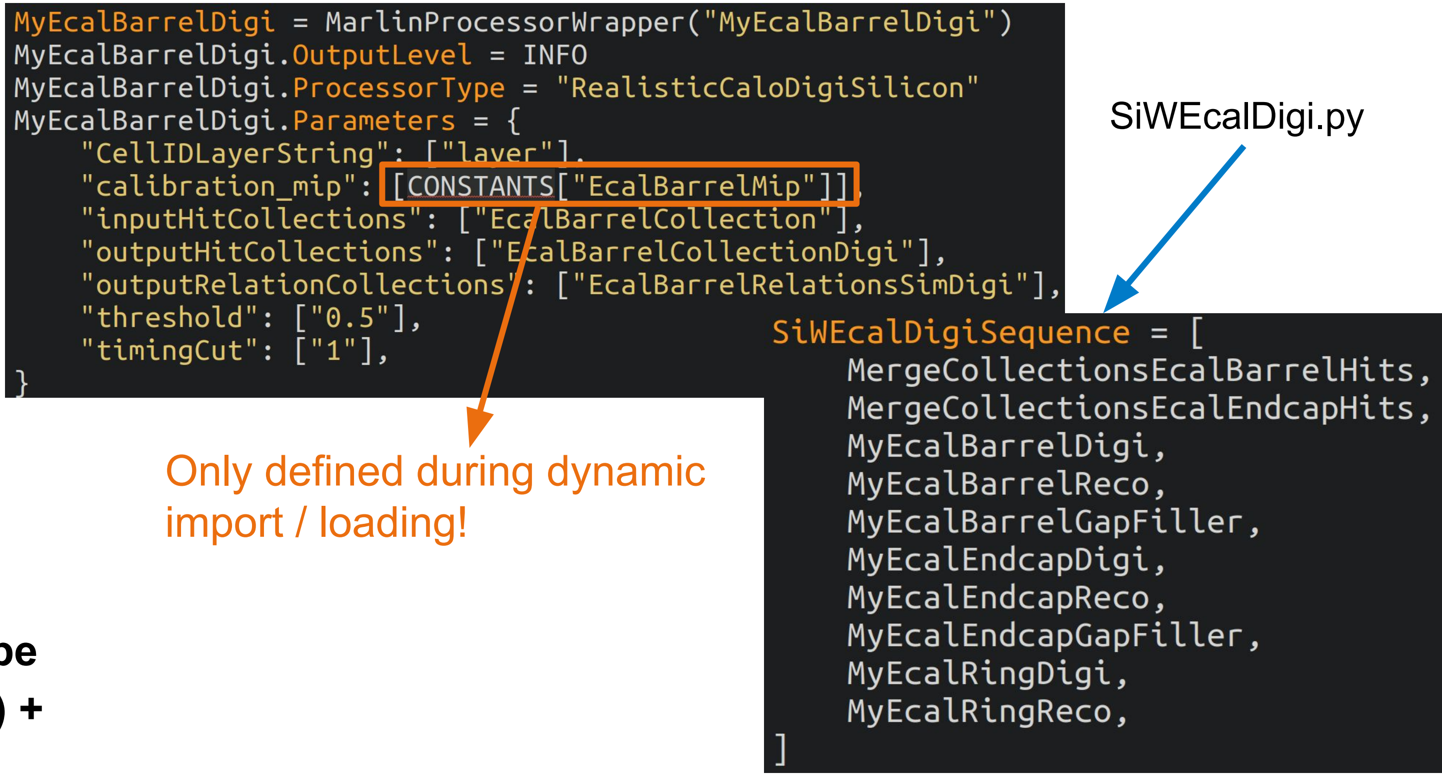


```
CONSTANTS = {
    "CMSEnergy": str(reco_args.cmsEnergy),
    "BeamCalCalibrationFactor": str(reco_args.beamCalCalibFactor),
}

det_calib_constants = import_from(
    f"Calibration/Calibration_{det_model}.cfg").CONSTANTS
CONSTANTS.update(det_calib_constants)

parseConstants(CONSTANTS)
```

New standard sequences

Plus a new convention

- Converted existing sequences (groups) from xml to python
  - Filenames: `s/.xml/.py/`
- All algorithms of a sequence go into a python list
  - New convention: **The name of this list has to be the same as the filename (without extension) + Sequence**
  - This defines the order of execution
- Provide helper functionality to facilitate dynamic inclusion
  - Also deals with passing in the configuration constants at load time
  - k4FWCore#178

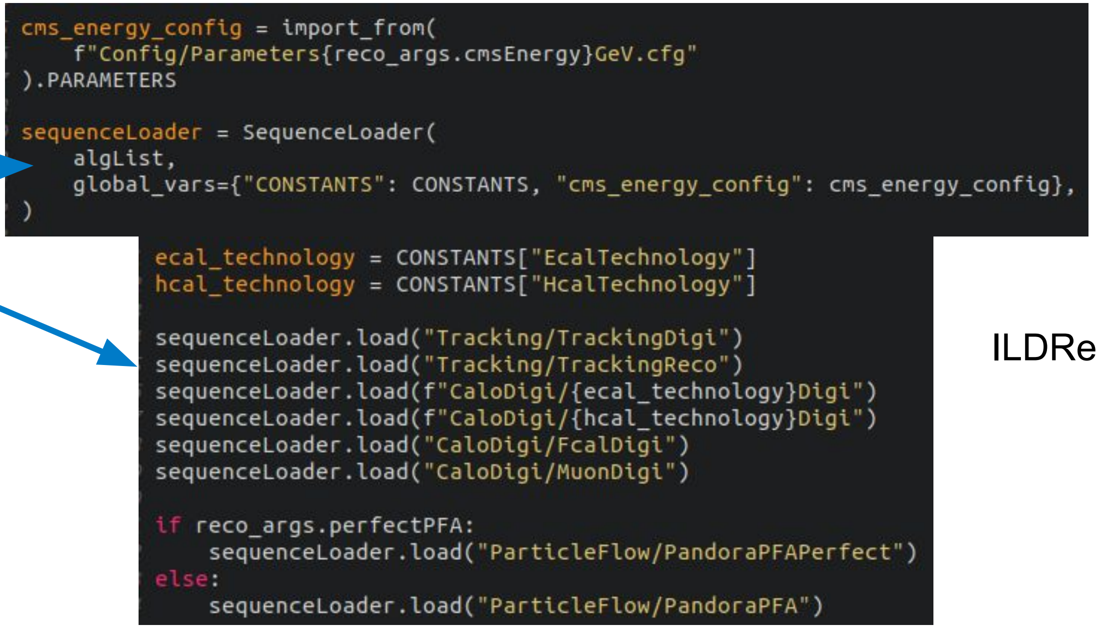


```
MyEcalBarrelDigi = MarlinProcessorWrapper("MyEcalBarrelDigi")
MyEcalBarrelDigi.OutputLevel = INFO
MyEcalBarrelDigi.ProcessorType = "RealisticCaloDigiSilicon"
MyEcalBarrelDigi.Parameters = {
    "CellIDLayerString": ["layer"],
    "calibration_mip": [CONSTANTS["EcalBarrelMip"]],
    "inputHitCollections": ["EcalBarrelCollection"],
    "outputHitCollections": ["EcalBarrelCollectionDigi"],
    "outputRelationCollections": ["EcalBarrelRelationsSimDigi"],
    "threshold": ["0.5"],
    "timingCut": ["1"],
}
```

Only defined during dynamic import / loading!

SiWEcalDigi.py

```
SiWEcalDigiSequence = [
    MergeCollectionsEcalBarrelHits,
    MergeCollectionsEcalEndcapHits,
    MyEcalBarrelDigi,
    MyEcalBarrelReco,
    MyEcalBarrelGapFiller,
    MyEcalEndcapDigi,
    MyEcalEndcapReco,
    MyEcalEndcapGapFiller,
    MyEcalRingDigi,
    MyEcalRingReco,
]
```



```
cms_energy_config = import_from(
    f"Config/Parameters{reco_args.cmsEnergy}GeV.cfg"
).PARAMETERS

sequenceLoader = SequenceLoader(
    algList,
    global_vars={"CONSTANTS": CONSTANTS, "cms_energy_config": cms_energy_config},
)
```

```
ecal_technology = CONSTANTS["EcalTechnology"]
hcal_technology = CONSTANTS["HcalTechnology"]

sequenceLoader.load("Tracking/TrackingDigi")
sequenceLoader.load("Tracking/TrackingReco")
sequenceLoader.load(f"CaloDigi/{ecal_technology}Digi")
sequenceLoader.load(f"CaloDigi/{hcal_technology}Digi")
sequenceLoader.load("CaloDigi/FcalDigi")
sequenceLoader.load("CaloDigi/MuonDigi")

if reco_args.perfectPFA:
    sequenceLoader.load("ParticleFlow/PandoraPFAPerfect")
else:
    sequenceLoader.load("ParticleFlow/PandoraPFA")
```

ILDReconstruction.py

How to run ILDReconstruction.py

The most important bit

- All arguments except input files have defaults; simplest case:

```
k4run ILDReconstruction.py --inputFiles=<your input files>
```

- Override the default detector model (ILD_l5_o1_v02) and CMS energy (250)

```
k4run ILDReconstruction.py --inputFiles=<your input files> \
      --cmsEnergy=500 \
      --detectorModel=ILD_s5_o4_v02
```

- Run current reconstruction through Gaudi (with default detector)

```
k4run ILDReconstrution.py --inputFiles=<your input files> \
      --lcioOutput=only
```

Summary & Next steps

- First version of modular Gaudi configuration for ILD standard reconstruction workflow
  - Possible to dynamically configure algorithms depending on calibration constants, etc.
  - Keeping existing functionality and basic concepts
  - Introduce a new convention
  - ILDConfig#137
- Develop a few python helpers to support dynamic imports
  - Directly upstream them to k4FWCore to make them available to everyone
  - k4FWCore#178
- Next steps:
  - Finalize all of the above (some details still need to be fixed) and make available
- Short term plans:
  - Integrate ILD_l5_v11 into this chain and run a first reconstruction
  - Convert miniDST workflow
- Mid term plans:
  - Make it possible to import standard sequences from “everywhere”

Thank you