Gaudi config for ILD reconstruction

The technical bits

Thomas Madlener, Feb 15 2024

Reminder: Marlin steering files basics

```
<marlin>
  <execute>   [1]
      ...  // the processors and processor groups to be execut
  </execute>

  <global>    [1]
      ...  // global parameter section
  </global>

  <processor> [n]
      ...  // definition of the processor and its parameters
  </processor>

  <group> [m]
      ...    // a group of processors
    <processor> [k]
      ...  // definition of the processor and its parameters
    </processor>
  </group>
</marlin>
```

- Several sections
  - `execute` and `global` are required
  - Others are optional but can also be repeated
  - Global section defines some global config parameters
  - Execute section defines the order of the processors

```
<global>
  <parameter name="LCIOInputFiles"> bbudsc_3evt_SIM.slcio </parameter>
  <parameter name="MaxRecordNumber" value="0"/>
  <parameter name="SkipNEvents" value="0"/>
  <parameter name="SupressCheck" value="false"/>
  <parameter name="Verbosity"> MESSAGE </parameter>
  <parameter name="RandomSeed" value="1234567890" />
</global>
```

Reminder: Marlin steering file constants

```
<constants>
  <constant name="CalibrationFactor"> 0.86 </constant>
  <constant name="CalibPath"> /home/toto/data/calib </constant>
  <constant name="YourParameter"> 42.0 </constant>
</constants>
```

- `constants` sections can be used to define arbitrary constants
- Constants can be referred to in many places in Marlin steering files
  - Other constants
  - Include references

```
<processor name="MyCalibrationProcessor" type="CalibrationProcessor">
  <parameter name="CalibrationFile" type="string"> ${CalibPath}/Calib_May.txt </parameter>
  <parameter name="Factor" type="float"> ${CalibrationFactor} </parameter>
</processor>
```

```
<!-- ***** Input files constants ***** -->
<!-- Geometry model dependant calibration co
<include ref="${CalibrationFile}" />
```

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

ILDConfig organization of steering files

- Top level `MarlinStdReco.xml`
  - Includes other steering and calibration files dynamically
- Several subfolders with dedicated configuration for parts of the reconstruction
  - Tracking, Calorimeters, ParticleFlow, HighLevelReco
- Detector dependent calibration files
  - Define geometry and technology specific constants
- CMS Energy dependent configuration files
- Some minor cross-referencing
  - Some calibration depends on geometry and energy

```
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_v02.xml
Calibration_ILD_s5_o1_v05.xml -> Calibration_ILD_s5_o1_v02.xml
Calibration_ILD_s5_o1_v04.xml -> Calibration_ILD_s5_o1_v02.xml
Calibration_ILD_s5_o1_v03.xml -> Calibration_ILD_s5_o1_v02.xml
Calibration_ILD_s5_o1_v02.xml
```

Reminder: convertMarlinToGaudi.py

- Converter script that takes a **parsed Marlin steering file** as input and that produces a Gaudi python options file
  - All includes and some of the constants are resolved already
  - Constants are dropped into a CONSTANTS dictionary
  - Not all constants are resolved
  - Recursive resolution of constants in constants
  - Python formatting from dictionary for replacement in parameters
- Produces one large python script with everything (~1300 LoC)

```
CONSTANTS = {
            'lcgeo_DIR':
"/home/tmadlener/work/.spack/spackages/k4geo/main/skylake-ubuntu22.04-gcc12.3.0/5r3keez/share/k4geo",
            'DetectorModel': "ILD_l5_o1_v02",
            'CompactFile': '%(lcgeo_DIR)s/ILD/compact/%(DetectorModel)s/%(DetectorModel)s.xml",
```

```
parseConstants(CONSTANTS)
```

```
In [1]: constants = {"c1": "some string", "c2": "another string"}

In [2]: "value of c1= %(c1)s" % constants
Out[2]: 'value of c1= some string'
```

```
InitDD4hep = MarlinProcessorWrapper("InitDD4hep")
InitDD4hep.OutputLevel = INFO
InitDD4hep.ProcessorType = "InitializeDD4hep"
InitDD4hep.Parameters = {
                         "DD4hepXMLFile": ["%(CompactFile)s" % CONSTANTS]
                         }
```

Required features from python

- Dynamic import of code
  - Filenames to import from depend on calibration constants
- Injection of global state during import
  - Configuration parameters of algorithms depend on calibration constants
  - Algorithm sequences to import and run depend on calibration constants (e.g. Calorimeter technology)
- Calibration constants themselves are dynamic
  - Effectively a large nested dictionary where we try to remove one level of nesting by splitting it into different files

Dynamically import code in python

1. Using `compile` and `exec`
    - Effectively loading the file verbatim into the current interpreter session
    - `compile` not strictly necessary but errors are nicer
2. Using `importlib` and related utilities
    - Effectively as if importing a module but using the path to the file (and arbitrary file endings)

Using `compile` and `exec`

```python
def load_file(opt_file: Union[str, os.PathLike]) -> None:
    """Load the file content and run it in the current
interpreter session"""
    with open(opt_file, "r") as file:
        code = compile(file.read(), file.name, "exec")
        exec(code, globals())


load_file("list.py")
print(l)
load_file("error.py")
```

```python
print("hello")

l = []
l.append(1)
l.append(2)
```

```python
d = {"a": 1, "b"}
```

```
hello
[1, 2]
Traceback (most recent call last):
  File "/home/tmadlener/work/playground/python/exec_load_file/main.py", line 16, in <module>
    load_file("error.py")
  File "/home/tmadlener/work/playground/python/exec_load_file/main.py", line 10, in load_file
    code = compile(file.read(), file.name, "exec")
  File "error.py", line 3
    d = {"a": 1, "b"}
                   ^
SyntaxError: ':' expected after dictionary key
```

Using `importlib` and related utils

Thomas + ChatGPT

```python
def import_from(
    filename: os.PathLike,
    module_name: Optional[str] = None,
    global_vars: Optional[Dict[str, Any]] = None,
) -> Any:
    """Dynamically imports a module from the specified file path."""
    module_name = module_name or os.path.basename(filename).replace(".", "_")
    loader = SourceFileLoader(module_name, filename)
    spec = importlib.util.spec_from_loader(loader.name, loader)
    module = importlib.util.module_from_spec(spec)

    if global_vars:
        module.__dict__.update(global_vars)

    loader.exec_module(module)
    return module


mod = import_from("globals.py", global_vars={"variable": 42})
print(mod.a_list)
```

```python
print(f"{variable=}")

a_list = [1, 2, 3]
```

```
variable=42
[1, 2, 3]
```

ILDReconstruction.py features

- argparse for argument parsing
  - Allows to catch some mistakes very early (e.g. non-existent calibration, …)
- Automatic input file format detection
  - Choose correct reader and inject potentially necessary conversion
- EDM4hep output by default
  - Effectively REC
  - LCIO output can be added via cl argos
- Some new conventions on how to modularize configuration

```python
from k4FWCore.parseArgs import parser

parser.add_argument(
    "--inputFiles",
    action="extend",
    nargs="+",
    metavar=["file1", "file2"],
    help="One or multiple input files",
)
parser.add_argument(
    "--compactFile",
    help="Compact detector file to use",
    default=f"{os.environ['K4GEO']}/ILD/compact/ILD_l5_v02/ILD_l5_v02.xml",
)
parser.add_argument(
    "--outputFileBase",
    help="Base name of all the produced output files",
    default="StandardReco",
)
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",
)

reco_args = parser.parse_known_args()[0]
```

New calibration

- Mirrored structure to existing XML calibration files
  - Converted using converter script
- Dynamically imported into ILDReconstruction.py
  - Before nested constants are parsed!
- Exactly the same mechanism for CMS energy dependent config

```
Calibration/Calibration_ILD_l4_o1_v02.cfg
Calibration/Calibration_ILD_l2_v02.cfg -> Calibration_ILD_l5_o2_v02.cfg
Calibration/Calibration_ILD_s5_o3_v02.cfg
Calibration/Calibration_ILD_s5_o2_v02.cfg
Calibration/Calibration_ILD_s5_o1_v06.cfg -> Calibration_ILD_s5_o1_v02.cfg
Calibration/Calibration_ILD_s5_o1_v05.cfg -> Calibration_ILD_s5_o1_v02.cfg
Calibration/Calibration_ILD_s5_o1_v04.cfg -> Calibration_ILD_s5_o1_v02.cfg
Calibration/Calibration_ILD_s5_o1_v03.cfg -> Calibration_ILD_s5_o1_v02.cfg
```

```
# Converted from Calibration_ILD_l5_o1_v02.xml
CONSTANTS = {
    "EcalBarrelMip": "0.0001575",
    "EcalEndcapMip": "0.0001575",
    "EcalRingMip": "0.0001575",
    "HcalBarrelMip": "0.0004925",
    "HcalEndcapMip": "0.0004725",
    "HcalRingMip": "0.0004875",
    "EcalBarrelEnergyFactors": ["0.0063520964756", "0.012902699188"],
    "EcalEndcapEnergyFactors": ["0.0067218419842", "0.013653744940"],
    "EcalRingEnergyFactors": ["0.0066536339", "0.0135151972"],
    "HcalBarrelEnergyFactors": "0.0287783798145",
    "HcalEndcapEnergyFactors": "0.0285819096797",
    "HcalRingEnergyFactors": "0.0349940637704",
    "MuonCalibration": "56.7",
    "PandoraEcalToMip": "153.846",
    "PandoraHcalToMip": "37.1747",
    "PandoraMuonToMip": "10.5263",
    "PandoraEcalToEMScale": "1.0",
    "PandoraHcalToEMScale": "1.0",
    "PandoraEcalToHadBarrelScale": "1.17344504717",
    "PandoraEcalToHadEndcapScale": "1.17344504717",
    "PandoraHcalToHadScale": "1.02821419758",
    "PandoraSoftwareCompensationWeights": [
        "1.59121",
        "-0.0281982",
        "0.000250616",
        "-0.0424222"
```

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

Configuring standard sequences

- Converted existing sequences (groups) from XML to python
  - Filenames: `s/.xml/.py/`
- All algorithms go into a python list
  - **New convention: The name of this list has to be the same as the filename (without extension) + Sequence**
- Provide helper to make dynamic inclusion simple

```
TrackingDigiSequence = [
    MySplitCollectionByLayer,
    VXDPlanarDigiProcessor_CMOSVXD5,
    SITPlanarDigiProcessor,
    FTDPixelPlanarDigiProcessor,
    FTDStripPlanarDigiProcessor,
    FTDDDSpacePointBuilder,
    SETPlanarDigiProcessor,
    SETDDSpacePointBuilder,
    MyTPCDigiProcessor,
]
```

TrackingDigi.py

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

add_sequence("Tracking/TrackingDigi", algList)
add_sequence("Tracking/TrackingReco", algList)
add_sequence(f"CaloDigi/{ecal_technology}Digi", algList)
add_sequence(f"CaloDigi/{hcal_technology}Digi", algList)
add_sequence("CaloDigi/FcalDigi", algList)
add_sequence("CaloDigi/MuonDigi", algList)
add_sequence("ParticleFlow/PandoraPFA", algList)
add_sequence("HighLevelReco/BeamCalReco", algList)
add_sequence("HighLevelReco/HighLevelReco", algList)
```

ILDReconstruction.py

Just one more wrapper, I promise

- Dynamic import of file from provided sequence name
  - Use previously defined `import_from` and inject necessary global calibration / config state
- Sequence list name is determined by filename
  - Dynamically get that from imported module
- Extend the algorithm list with new algorithms

```python
def add_sequence(sequence: str, alg_list: list):
    """Add a sequence to the list of algorithms."""
    filename = f"{sequence}.py"
    seq_name = f"{sequence.split('/')[-1]}Sequence"

    seq_module = import_from(
        filename,
        global_vars={"CONSTANTS": CONSTANTS, "cms_energy_config": cms_energy_config},
    )
    seq = getattr(seq_module, seq_name)
    alg_list.extend(seq)
```

Summary

- k4FWCore#178 for python helpers
  - `load_file` and `import_from`
- ILDConfig#137 for new Gaudi based configuration
- First version of modular Gaudi configuration for ILD standard reco
  - Possible to dynamically configure algorithms depending on calibration constants, etc..
  - Keep all existing functionality (🤞)
- ChatGPT is pretty useful
  - import_from took me 10 mins (probably ~1 hour without)
  - Generated documentation is usable with minor adjustments