

Python bindings for ACTS Examples

Paul Gessinger

CERN

15.06.2021 - ACTS Developer Meeting



Wait... what???

Examples structure as it stands

- We have an $M \times N$ setup:

		Geometry		
		DD4hep+Propagation	TGeo+Propagation	Generic+Propagation
		DD4hep+Digitization	TGeo+Digitization	Generic+Digitization
		DD4hep+Seeding	TGeo+Seeding	Generic+Seeding
Algorithm		DD4hep+Track Finding	TGeo+Track Finding	Generic+Track Finding
		DD4hep+Track Fitting	TGeo+Track Fitting	Generic+Track Fitting

- Mitigation: factorize Geometry and Algorithms, *thin* executables
 - ▶ We have **a lot** of executables (>50)
- Have command line options (boost) to configure
 - ▶ Reused between example types
 - ▶ **Many** options, because there is only one namespace
 - ▶ Default values **are split** between Config struct and option declaration
- Which options exist where is hard to remember
- Documentation how to run examples goes out of date quickly

Example: Particle generation

```
bin/ActsExampleParticleGun \
--events=100 \
--output-dir=data/gen/four_muons \
--output-csv \
--gen-phi-degree=0:90 \
--gen-eta=-2:2 \
--gen-mom-gev=1:5 \
--gen-pdg=13 \
--gen-randomize-charge \
--gen-nparticles=4
```

```
bin/ActsExamplePythia8 \
--events=100 \
--output-dir=data/gen/ttbar_mu140 \
--output-csv \
--rnd-seed=42 \
--gen-cms-energy-gev=14000 \
--gen-hard-process=Top:qqbar2ttbar=on \
--gen-npileup=140
```

Program option help I

```
bin/ActsExamplePropagationDD4hep -h
-h [ --help ]
-l [ --loglevel ] arg (=2)
--response-file arg
-n [ --events ] arg
--skip arg (=0)
-j [ --jobs ] arg (=-1)
--geo-surface-loglevel arg (=3)
--geo-layer-loglevel arg (=3)
--geo-volume-loglevel arg (=3)
--mat-input-type arg (build)
--mat-input-file arg
--mat-output-file arg
--mat-output-sensitives arg (=1)

Produce help message
The output log level. Please set the
wished number (0 = VERBOSE, 1 = DEBUG,
2 = INFO, 3 = WARNING, 4 = ERROR, 5 =
FATAL).
Configuration file (response file)
replacing command line options.
The number of events to process. If not
given, all available events will be
processed.
The number of events to skip
Number of parallel jobs, negative for
automatic.
The output log level for the surface
building.
The output log level for the layer
building.
The output log level for the volume
building.
The way material is loaded: 'none',
'build', 'proto', 'file'.
Name of the material map input file,
supported: '.json' or '.root'.
Name of the material map output file
(without extension).
Write material information of sensitive
```

Program option help II

--mat-output-approaches arg (=1)	surfaces. Write material information of approach surfaces.
--mat-output-representing arg (=1)	Write material information of representing surfaces.
--mat-output-boundaries arg (=1)	Write material information of boundary surfaces.
--mat-output-volumes arg (=1)	Write material information of volumes.
--mat-output-dense-volumes arg (=0)	Write material information of dense volumes.
--mat-output-allmaterial arg (=0)	Add protoMaterial to all surfaces and volume for the mapping.
--bf-constant-tesla arg	Set a constant magnetic field vector in Tesla. If given, this takes preference over all other options.
--bf-scalable	If given, the constant field strength will be scaled differently in every event. This is for testing only.
--bf-scalable-scalor arg (=1.25)	Scaling factor for the event-dependent field strength scaling. A unit value means that the field strength stays the same for every event.
--bf-map-file arg	Read a magnetic field map from the given file. ROOT and text file formats are supported. Only used if no constant field is given.

Program option help III

--bf-map-tree arg (=bField)	Name of the TTree in the ROOT file. Only used if the field map is read from a ROOT file.
--bf-map-type arg (=xyz)	Either 'xyz' or 'rz' to define the type of the field map. If given, the field map is assumed to describe only the first octant/quadrant and the field is symmetrically extended to the full space.
--bf-map-octantonly	
--bf-map-lengthscale-mm arg (=1)	Optional length scale modifier for the field map grid. This option only needs to be set if the length unit in the field map file is not 'mm'. The value must scale from the stored unit to the equivalent value in 'mm'.
--bf-map-fieldscale-tesla arg (=1)	Optional field value scale modifier for the field map value. This option only needs to be set if the field value unit in the field map file is not 'Tesla'. The value must scale from the stored unit to the equivalent value in 'Tesla'.
--bf-solenoid-mag-tesla arg (=0)	The magnitude of a solenoid magnetic field in the center in 'Tesla'. Only used if neither constant field nor a magnetic field map is given.
--bf-solenoid-length arg (=6000)	The length of the solenoid magnetic

Program option help IV

```
--bf-solenoid-radius arg (=1200)           field in `mm`.  
--bf-solenoid-ncoils arg (=1194)           The radius of the solenoid magnetic  
                                            field in `mm`.  
--bf-solenoid-map-rlim MIN:MAX (=0:1200)    Number of coils for the solenoid  
                                            magnetic field.  
--bf-solenoid-map-zlim MIN:MAX (= -3000:3000) The length bounds of the grid created  
                                            from the analytical solenoid field in  
`mm`.  
--bf-solenoid-map-nbins arg (=150:200)       The radius bounds of the grid created  
                                            from the analytical solenoid field in  
`mm`.  
--rnd-seed arg (=1234567890)                The number of bins in r-z directions  
                                            for the grid created from the  
analytical solenoid field.  
--prop-debug arg (=0)                       Random numbers seed.  
--prop-step-collection arg (=propagation-steps) Run in debug mode, will create  
                                            propagation screen output.  
--prop-stepper arg (=1)                      Propagation step collection.  
--prop-mode arg (=0)                        Propagation type: 0 (StraightLine), 1  
                                            (Eigen), 2 (Atlas).  
                                            Propagation modes: 0 (inside-out), 1  
                                            (surface to surface).
```

Program option help V

--prop-cov arg (=0)	Propagate (random) test covariances.
--prop-energyloss arg (=1)	Apply energy loss correction - in extrapolation mode only.
--prop-scattering arg (=1)	Apply scattering correction - in extrapolation mode only.
--prop-record-material arg (=1)	Record the material interaction and - in extrapolation mode only.
--prop-material-collection arg (=propagation-material)	Propagation material collection.
--prop-ntests arg (=1000)	Number of tests performed.
--prop-resolve-material arg (=1)	Resolve all smaterial surfaces.
--prop-resolve-passive arg (=0)	Resolve all passive surfaces.
--prop-resolve-sensitive arg (=1)	Resolve all sensitive surfaces.
--prop-d0-sigma arg (=0.01499999999999999)	Sigma of the transverse impact parameter [in mm].
--prop-z0-sigma arg (=55)	Sigma of the longitudinal impact parameter [in mm].
--prop-phi-sigma arg (=0.001)	Sigma of the azimuthal angle [in rad].
--prop-theta-sigma arg (=0.001)	Sigma of the polar angle [in rad].
--prop-qp-sigma arg (=0.0001)	Sigma of the signed inverse momentum [in GeV^{-1}].
--prop-t-sigma arg (=1)	Sigma of the time parameter [in ns].
--prop-corr-offd arg	The 15 off-diagonal correlation $\rho(d_0, z_0)$, $\rho(d_0, \phi)$, [...], $\rho(z_0, \phi)$, $\rho(z_0, \theta)$, [...],

Program option help VI

```
rho(qop,t). Row-wise.  
--prop-phi-range arg (=-3.14159:3.14159)      Azimuthal angle phi range for  
                                                proprapolated tracks.  
--prop-eta-range arg (=-4:4)                    Pseudorapidity range for proprapolated  
                                                tracks.  
--prop-pt-range arg (=0.1:100)                  Transverse momentum range for  
                                                proprapolated tracks [in GeV].  
--prop-max-stepsizE arg (=3000)                Maximum step size for the propagation  
                                                [in mm].  
--prop-pt-loopers arg (=0.5)                   Transverse momentum below which loops  
                                                are being detected [in GeV].  
--output-dir arg                                Output directory location.  
--output-root                                     Switch on to write '.root' output  
                                                file(s).  
--output-obj                                      Switch on to write '.obj' ouput  
                                                file(s).  
--dd4hep-input arg                             The locations of the input DD4hep  
                                                files, use 'file:foo.xml'. In case you  
want to read in multiple files, add the  
option multiple times.  
--dd4hep-envelopeR arg (=1)                   The envelop cover in R for DD4hep  
                                                volumes in mm.  
--dd4hep-envelopeR arg (=1)                   The tolerance added to the geometrical  
extension in r of the layers contained  
to build the volume envelope around in
```

Program option help VII

```
--dd4hep-envelopeZ arg (=1)           mm.  
                                         The tolerance added to the geometrical  
                                         extension in z of the layers contained  
                                         to build the volume envelope around in  
                                         mm.  
--dd4hep-layerThickness arg (=1.0000000000000001e-09)  
                                         In case no surfaces (to be contained by  
                                         the layer) are handed over, the layer  
                                         thickness will be set to this value.  
--dd4hep-buildFCChh arg (=1)  
                                         If you are not building the FCChh  
                                         detector please set this flag to false.  
--dd4hep-loglevel arg (=2)  
                                         The output log level of the geometry  
                                         building. Please set the wished number  
                                         (0 = VERBOSE, 1 = DEBUG, 2 = INFO, 3 =  
                                         WARNING, 4 = ERROR, 5 = FATAL).
```

Tons of program options

- WHY? Because the examples are slow to compile!
 - ▶ Nobody wants to wait every time they tweak their parameters
- To mitigate the number of options, we have *response files*
 - ▶ Write options into file, parse the file
- So effectively, we have a configuration language already, but a terrible one

So... Python!

So... Python!

What the idea is:

Bindings for the **Examples** code (Sequencer, RNG, Readers, Writers, Algorithms)

What the idea is NOT:

Bindings for anything we don't strictly need for the above (i.e. no tracking code in Python)

Demonstrator

- I built a demonstrator using pybind11 to see what this would look like
- Don't change semantics, but a bit of sugar to make it more pythonic
- SomeAlgorithm::Config is the **canonical** source for defaults

```
class SomeAlgorithm : public BareAlgorithm {  
public:  
    struct Config {  
        double valueA{5};  
        double valueB{8};  
        std::string name{"nothing"};  
        std::vector<std::string> extra{};  
    };  
  
    SomeAlgorithm(const Config& cfg, Acts::Logging::Level level);  
};
```

Demonstrator

- I built a demonstrator using pybind11 to see what this would look like
- Don't change semantics, but a bit of sugar to make it more pythonic
- SomeAlgorithm::Config is the **canonical** source for defaults

C++

```
SomeAlgorithm::Config cfg;
cfg.valueA = 42;
cfg.valueB = 99;
cfg.name = "some_alg";
cfg.extra
    = {{"some", "extra", "values"}};
auto alg
    = std::make_shared<SomeAlgorithm>(
        cfg, Acts::Logging::INFO);
sequencer.addAlgorithm(alg);
```

CPPython

```
acts.examples.SomeAlgorithm.Config cfg
cfg.valueA = 42
cfg.valueB = 99
cfg.name = "some_alg"
cfg.extra
    = ["some", "extra", "values"]
alg = acts.examples.SomeAlgorithm(
    cfg, acts.logging.INFO)
sequencer.addAlgorithm(alg)
```

Demonstrator

- I built a demonstrator using pybind11 to see what this would look like
- Don't change semantics, but a bit of sugar to make it more pythonic
- SomeAlgorithm::Config is the **canonical** source for defaults

C++

```
SomeAlgorithm::Config cfg;
cfg.valueA = 42;
cfg.valueB = 99;
cfg.name = "some_alg";
cfg.extra
= {{"some", "extra", "values"}};
auto alg
= std::make_shared<SomeAlgorithm>(
    cfg, Acts::Logging::INFO);
sequencer.addAlgorithm(alg);
```

Python

```
alg = acts.examples.SomeAlgorithm(
    valueA = 42,
    valueB = 99,
    name = "some_alg",
    extra = ["some", "extra", "values"],
    level = acts.logging.INFO
)
sequencer.addAlgorithm(alg)
```

Seeding example

Seeding example I

```
import acts
import acts.examples
u = acts.UnitConstants

detector, trackingGeometry, _ = acts.examples.GenericDetector.create()
# OR
detector, trackingGeometry, _ = acts.examples.DD4hepDetector.create(
    xmlFileNames=["thirdparty/OpenDataDetector/xml/OpenDataDetector.xml"])
)

field = acts.ConstantBField(acts.Vector3(0, 0, 2 * u.T))
# OR
field = acts.SolenoidBField(radius=1.2*u.m, length=6*u.m,
                            bMagCenter=2*u.T, nCoils=1194)
rnd = acts.examples.RandomNumbers()
```

Seeding example II

```
evGen = acts.examples.EventGenerator( # Particle gun
    level=acts.logging.INFO,
    generators=[
        acts.examples.EventGenerator.Generator(
            multiplicity=acts.examples.FixedMultiplicityGenerator(n=2),
            vertex=acts.examples.GaussianVertexGenerator(
                stddev=acts.Vector4(0, 0, 0, 0), mean=acts.Vector4(0, 0, 0, 0)
            ),
            particles=acts.examples.ParametricParticleGenerator(
                p=(1 * u.GeV, 10 * u.GeV), eta=(-2, 2), phi=(0, 90 * u.degree),
                randomizeCharge=True, numParticles=4,
            ),
        )
    ],
    outputParticles="particles_input", randomNumbers=rnd
)
```

Seeding example III

```
# OR: Read input from input collection (e.g. Pythia8 output)

particlesIn = acts.examples.RootParticleReader(
    level=acts.logging.INFO,
    particleCollection="particles_input",
    inputDir="output",
    inputFile="pythia8_particles.root",
)
```

Seeding example IV

```
# Simulation
simAlg = acts.examples.FatrasAlgorithm(
    level=acts.logging.INFO,
    inputParticles=evGen.config.outputParticles, # automatically consistent
    # OR: inputParticles=particlesIn.config.particleCollection,
    outputParticlesInitial="particles_initial",
    outputParticlesFinal="particles_final",
    outputSimHits="simhits",
    randomNumbers=rnd,
    trackingGeometry=trackingGeometry,
    magneticField=field,
    generateHitsOnSensitive=True,
)
```

Seeding example V

```
# Digitization
digiCfg = acts.examples.DigitizationConfig(
    acts.examples.readDigiConfigFromJson(
        "Examples/Algorithms/Digitization/share/"
        "default-smearing-config-generic.json"
    )
)
digiCfg.trackingGeometry = trackingGeometry
digiCfg.randomNumbers = rnd
digiCfg.inputSimHits = simAlg.config.outputSimHits
digiAlg = acts.examples.DigitizationAlgorithm(digiCfg, acts.logging.INFO)
```

Seeding example VI

```
# Space point formation
spAlg = acts.examples.SpacePointMaker(
    level=acts.logging.INFO,
    inputSourceLinks=digiCfg.outputSourceLinks,
    inputMeasurements=digiCfg.outputMeasurements,
    outputSpacePoints="spacepoints",
    trackingGeometry=trackingGeometry,
    geometrySelection=acts.examples.readJsonGeometryList(
        "Examples/Algorithms/TrackFinding/share/geoSelection-genericDetector.json"
    ),
)
```

Seeding example VII

```
# Seeding
seedingAlg = acts.examples.SeedingAlgorithm(
    level=acts.logging.INFO,
    inputSpacePoints=[spAlg.config.outputSpacePoints],
    outputSeeds="seeds",
    outputProtoTracks="prototracks",
)
```

Seeding example VIII

```
# Parameter estimation
parEstimateAlg = acts.examples.TrackParamsEstimationAlgorithm(
    level=acts.logging.INFO,
    inputProtoTracks=seedingAlg.config.outputProtoTracks,
    inputSpacePoints=[spAlg.config.outputSpacePoints],
    inputSourceLinks=digiCfg.outputSourceLinks,
    outputTrackParameters="estimatedparameters",
    outputProtoTracks="prototracks_estimated",
    trackingGeometry=trackingGeometry,
    magneticField=field,
)
```

Seeding example IX

```
tfPerf = acts.examples.TrackFinderPerformanceWriter(  
    level=acts.logging.INFO,  
    inputProtoTracks=seedingAlg.config.outputProtoTracks,  
    inputParticles=simAlg.config.outputParticlesFinal,  
    inputMeasurementParticlesMap=digiCfg.outputMeasurementParticlesMap,  
    outputDir="output",  
    outputFilename="performance_seeding_trees.root",  
)  
  
seedPerf = acts.examples.SeedingPerformanceWriter(  
    level=acts.logging.INFO,  
    inputProtoTracks=seedingAlg.config.outputProtoTracks,  
    inputParticles=simAlg.config.outputParticlesFinal,  
    inputMeasurementParticlesMap=digiCfg.outputMeasurementParticlesMap,  
    outputDir="output",  
    outputFilename="performance_seeding_trees.root",  
)
```

Seeding example X

```
tpWriter = acts.examples.RootTrackParameterWriter(  
    level=acts.logging.INFO,  
    inputTrackParameters=parEstimateAlg.config.outputTrackParameters,  
    inputProtoTracks=parEstimateAlg.config.outputProtoTracks,  
    inputParticles=simAlg.config.outputParticlesFinal,  
    inputSimHits=simAlg.config.outputSimHits,  
    inputMeasurementParticlesMap=digiCfg.outputMeasurementParticlesMap,  
    inputMeasurementSimHitsMap=digiCfg.outputMeasurementSimHitsMap,  
    outputDir="output",  
    outputFilename="estimatedparams.root",  
    outputTreename="estimatedparams",  
)
```

Seeding example XI

```
# Set up sequencer and run the job
s = acts.examples.Sequencer(events=100, logLevel=acts.logging.INFO)

s.addReader(evGen)
s.addAlgorithm(simAlg)
s.addAlgorithm(digiAlg)
s.addAlgorithm(spAlg)
s.addAlgorithm(seedingAlg)
s.addAlgorithm(parEstimateAlg)
s.addWriter(tfPerf)
s.addWriter(seedPerf)
s.addWriter(tpWriter)

s.run() # Blocks until we're done
```

Seeding example XII

```
# or add some more in between output
s.addWriter(acts.examples.RootParticleWriter(
    level=acts.logging.INFO,
    inputParticles=evGen.config.outputParticles,
    filePath="output/evgen_particles.root",
))
s.addWriter(acts.examples.RootParticleWriter(
    level=acts.logging.INFO,
    inputParticles=simAlg.config.outputParticlesFinal,
    filePath="output/fatras_particles_final.root",
))
s.addWriter(acts.examples.RootParticleWriter(
    level=acts.logging.INFO,
    inputParticles=simAlg.config.outputParticlesInitial,
    filePath="output/fatras_particles_initial.root",
))
```

How to proceed?

- Subset of Examples API currently implemented
- Binding code is ~ straightforward
- Structural choice: toggle Python bindings via CMake flag, options:
 1. `-DACTS_BUILD_PLUGIN_PYTHON=ON` and the Python plugin checks if, e.g. DD4hep is enabled (this is what I'm currently doing)
 2. `-DACTS_BUILD_PYTHON_BINDINGS=ON` and e.g. the DD4hep plugin checks if Python bindings are enabled
- Ultimately we **could** replace the example binaries with scripts like `seeding.py`
- Thoughts?

Backup