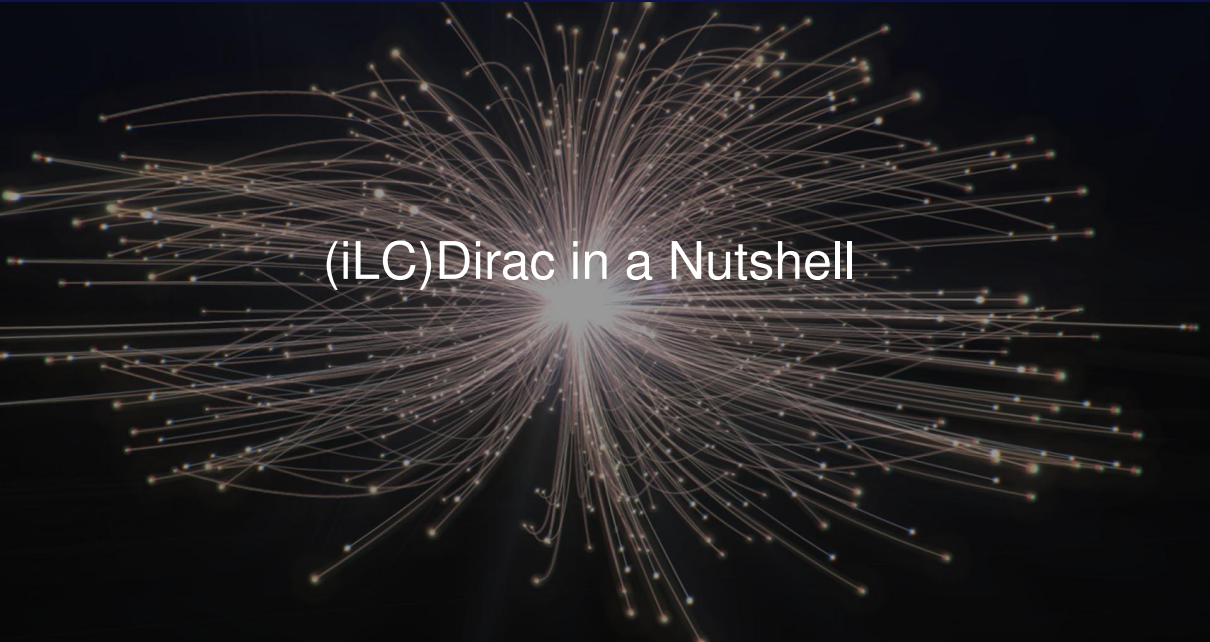
Tools for MC Production
The international Lepton Collider Dirac Instance (iLCDirac)

André Sailer

CERN-EP-SFT

FCC Physics Workshop
January 24, 2023

Table Of Contents

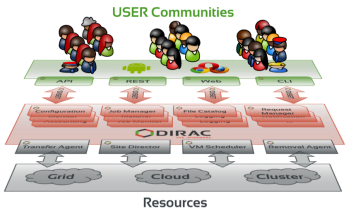


(iLC)Dirac in a Nutshell

Dirac in a Nutshell

iLCDirac is based on the DIRAC interware originally developed for LHCb

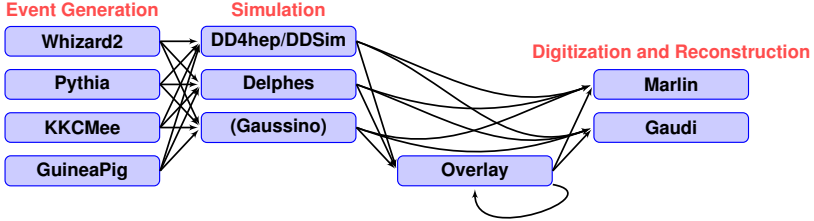
- Dirac (Distributed Infrastructure with Remote Agent Control): High level interface between users and distributed resources
- Distributed Workload Management: one interface to execute anywhere: batch farms, grid computing elements, HPCs
- Data Management (file transfers, meta data augmented file catalog)
- High degree of automation
- Web interface for controlling jobs



diracgrid.org

iLCDirac Use Cases

- The iLCDirac extension of Dirac is set up for the ILC, Calice, and FCC Virtual Organisations
- Centralized MC Production (Event Generation, Geant4 Simulation, Reconstruction)
- User jobs (Generation, Simulation, Reconstruction, Analyses)
- iLCDirac uses almost all functionality provided by DIRAC
- Specific features in iLCDirac
  - Workflow Modules for Key4hep Software (see later)



  - Overlay System for adding beam background files to MC jobs efficiently and effectively
  - Pandora Particle Flow calibration service (Marlin based)

Source Code: **https://gitlab.cern.ch/CLICdp/ILCDIRAC**

Requirements

- Member of the FCC Virtual Organisation
- An iLCDirac client and a compatible operating system (linux or macOS)
  - Centrally installed client on CVMFS
- Minimal knowledge of python

Application Examples

Job Example: MC Generation

```
from DIRAC.Core.Base import Script
Script.parseCommandLine()

from DIRAC import gLogger

from DIRAC.Resources.Catalog.FileCatalog import FileCatalog
from ILCDIRAC.Interfaces.API.DiracILC import DiracILC
from ILCDIRAC.Interfaces.API.NewInterface.UserJob import UserJob
from ILCDIRAC.Interfaces.API.NewInterface.Applications import KKMC

job = UserJob()

kkmc = KKMC ()
kkmc.setVersion('LCG_97a_FCC_4')
kkmc.setEvtType('Mu')
kkmc.setEnergy(91.2)
kkmc.setNumberOfEvents(1000)
kkmc.setOutputFile('kkmu_1000.LHE')

job.append(kkmc)
job.submit(dIlc)
```

Job Example: Particle Gun Simulation

```
outputFiles = ['electron.slcio', 'muon.slcio', 'pion.slcio']
particles = ['e-', 'mu-', 'pi-']
job = UserJob()
job.setName('DDSimTest_%n')
job.setSplitDoNotAlterOutputFilename()
job.setSplitParameter('particle', particles)
job.setSplitParameter('outputFile', outputFiles)
job.setSplitOutputData(outputFiles, 'test/ddsim', 'CERN-DST-EOS')

ddsim = DDSim()
ddsim.setVersion('ILCSoft-2018-08-10_gcc62')
ddsim.setDetectorModel('CLIC_o3_v14')

# the named placeholder '%(particle)s' has the same name as the first argument of setSplitParamet
ddsim.setExtraCLIArguments('--gun.particle=%(particle)s')
ddsim.setOutputFile('%(outputFile)s.slcio')
```

Should work for any string based application parameter, let me know your use case.

Job Example: Reconstruction

```
1  from DIRAC.Resources.Catalog.FileCatalog import FileCatalog
2  import DiracILC, UserJob, GaudiApp # incomplete for lack of space
3
4  dIlc = DiracILC()
5
6  inputData = FileCatalog().findFilesByMetadata({'ProdID': 23456, 'DataType': 'SIM'})
7  inputData = inputData['Value']  # assuming success
8
9  job = UserJob()
10 job.setName("SplitTest_%n")  # %n will be replaced by the task number
11 job.setFCCConfig('key4hep-latest')
12 job.setOutputData("RecoTest.slcio", OutputPath="RecoTest")
13 job.setSplitInputData(inputData, numberOfFilesPerJob=10)
14
15 gaudi = GaudiApp()
16 gaudi.setVersion('key4hep-latest')
17 gaudi.setSteeringFile('fcc_e4h_reco.py')
18 gaudi.setOutputFile('RecoTest.root')
19
20 job.append(ga); job.submit(dIlc)
```

Line 13 is the key, *inputFile* will be automatically filled for the application. Faster job submission.

Transformations: Centralized MC Productions

The Transformations

Job splitting is nice, but for central MC Productions more powerful features are available.

- Run some MC generator, followed by Geant4 Simulation of a detector, followed by reconstruction
- Do not want to wait for one stage to finish before starting with the next
- Enter the “Transformation” system: A set of tools that automatically creates new tasks when needed
- We only have to define the workflows and the number of desired primary tasks, which can be extended later on as well.

Transformation Types

- A *DataProcessing* transformation runs applications to create output data (we will focus on this)
  - e.g.: MCGeneration, Simulation, Reconstruction, Stripping/Merging
  - To set up define the application(s) similarly to a job as before
  - The *Transformation System* in DIRAC takes care of job creation, resubmission, finding input files
  - Plugins to the system allow one to, for example, configure what an outputfile should look like, how the grouping should be done, where to send the jobs
- A *DataManipulation* replicate or move files around
  - Again the transformation system takes care of (re)submission for given list of files, until all tasks successful or error threshold is reached

DataProcessing Script in iLCDirac

- Script sets workflow module parameters (Steering file, detector model, etc.)
- Reads 'conf' file, creates chain of given *ProdTypes*, for each *process* or *prodID*
- Metadata used to define input data (EventType, ProdID) and metadata for output folders
- Productions implicitly coupled. TransformationID part of input data query for next step
- Config file on the right: create 3 transformations of KKMC followed by Delphes for different energies

```
# [snip]
configVersion = key4hep-devel
configPackage = fccConfig
eventsPerJobs = 1000
detectorModel = IDEA
generatorApplication = KKMC
generatorVersion = key4hep_nightly
generatorSteeringFile =
delphesAfterGen = True

# can define list of energies or processes
# ("scalar" product)
energies = 91.2, 150, 300
processes = mumu, mumu, mumu

# different workflows as different prod types
ProdTypes = Gen
# [snip]
```

DataManipulation Transformations

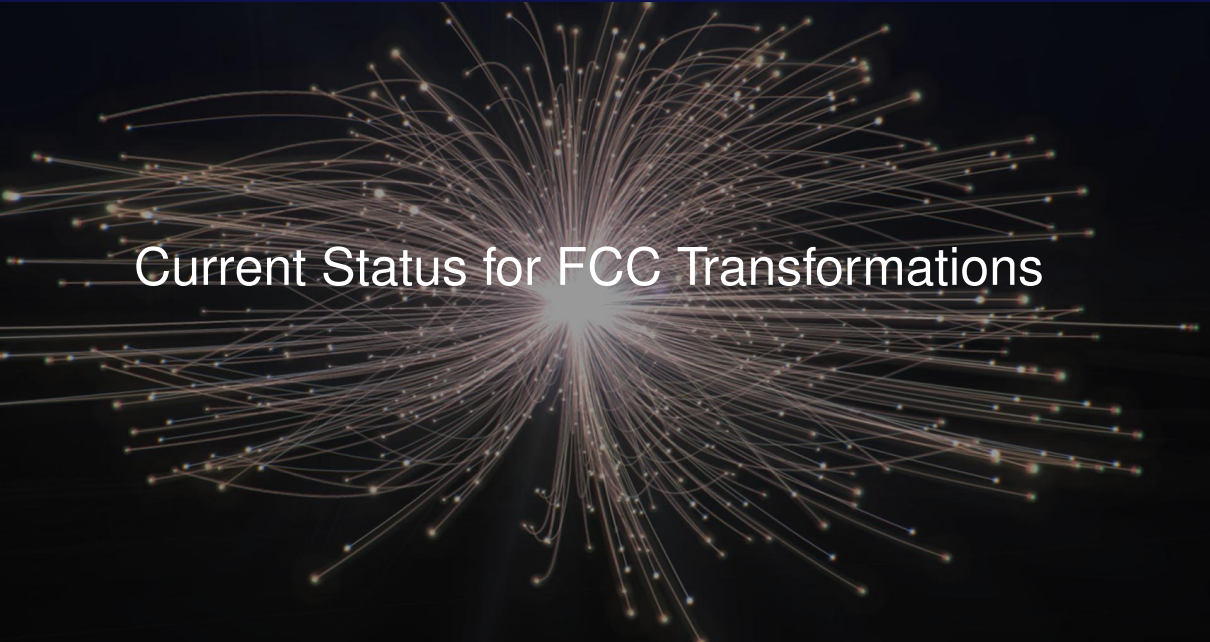
- scripts to define replication (staging) or moving transformations
  - Replicate files from A to B, optionally remove from A
  - Uses FTS3
- Set ProdID, SourceSE, TargetSE, Datatype; optional: GroupSize
- **dirac-ilc-replication-transformation**
- **dirac-ilc-moving-transformation**

Consistency Checks: DataRecoveryAgent, FileStatusTransformationAgent
Here be Dragons

Source

- DataRecoveryAgent: Checks all production jobs for consistency, treat inconsistent states
  - All output files present for successful jobs, remove output files that shouldn't exist, input file exists, one input file has only one set of descendants,. . .
- FileStatusTransformationAgent: checks consistency of *DataManipulation* transformations, reset *Requests*
  - files exist at destinations, files exist at all

Nothing productions managers have to worry about, but they are there to watch your back!



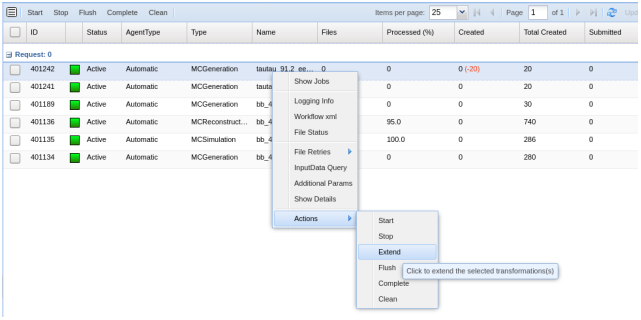
Current Status for FCC Transformations

How to create the transformation

- `source /cvmfs/clicdp.cern.ch/DIRACPy3/bashrc`
- `dirac-proxy-init -g fcc_prod` # for the privileged
- `dirac-fcc-make-productions -p > configFile` # modify
- `dirac-fcc-make-productions -f configFile`
- Most workflows are still work in progress! (Technical Student starting next Month)

Need more files

WebInterface



or

```
dirac-ilc-add-tasks-to-prod.py 412345 30
```

Find the files, production info

```
dirac-dms-find-lfns Path=/ ProdID=412345 Datatype=delphes
dirac-dms-find-lfns Path=/ EvtType=tautau Energy=91.2 Datatype=delphes \
      DetectorType=IDEA
dirac-ilc-get-info -p 412345
dirac-ilc-get-prod-log -P 412345
```

Transformation Metadata Example

- Metadata set at creation of transformation

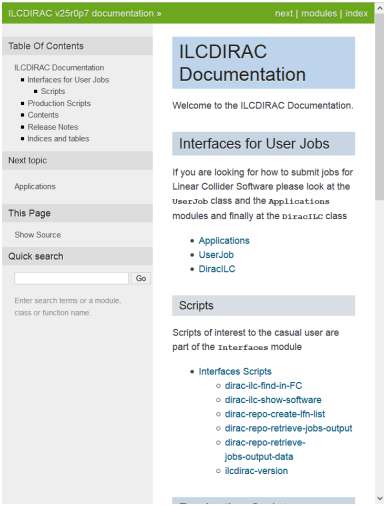
```
/fcc/ee/91.2gev/: {'Energy': '91.2'}
/fcc/ee/91.2gev/tautau/: {'EvtType': 'tautau'}
/fcc/ee/91.2gev/tautau/kkmcee: {'Generator': 'kkmcee'}
/fcc/ee/91.2gev/tautau/kkmcee/delphes: {'Datatype': 'delphes'}
/fcc/ee/91.2gev/tautau/kkmcee/delphes/IDEA: {'DetectorType': 'IDEA'}
/fcc/ee/91.2gev/tautau/kkmcee/delphes/IDEA/00012345: {'NumberOfEvents': 1000, 'ProdID': 1
/fcc/ee/91.2gev/tautau/kkmcee/lhef: {'Datatype': 'lhef'}
/fcc/ee/91.2gev/tautau/kkmcee/lhef/00012345: {'NumberOfEvents': 1000, 'ProdID': 12345}
Setting non searchable metadata information:
  /fcc/ee/91.2gev/tautau/kkmcee/delphes/IDEA/00012345:
  {'SWPackages': 'kkmc.key4hep_nightly;gaudiapp.key4hep-latest'}
Setting non searchable metadata information:
  /fcc/ee/91.2gev/tautau/kkmcee/lhef/00012345:
  {'SWPackages': 'kkmc.key4hep_nightly'}
```

- Some metadata can also be written for each file (but this is less efficient for the backend system)

Documentation

Documentation

- http://lcd-data.web.cern.ch/lcd-data/doc/ilcdiracdoc/
- Information about commands (scripts) including options
- API, examples for all applications



Support

- In case of fire:
  1. Consult documentation:
     `http://lcd-data.web.cern.ch/lcd-data/doc/ilcdiracdoc/`
  2. Before submitting a ticket, see: `http://lcd-data.web.cern.ch/lcd-data/doc/ilcdiracdoc/DOC/Files/UserGuide/support.html`
  3. Submit a ticket to the issue tracker
     `https://its.cern.ch/jira/browse/ILCDIRAC`
     - See also “Report a Problem” buttons in web portal and documentation
  4. Email: ilcdirac-support@cern.ch

Summary

Summary

- iLCDirac is an effective tool for distributed computing: workload management, data management
- Ergonomic support for centralized productions
  - Simple interface to create productions
  - Automatic transformation system to do the rest
- More FCC workflows will be implemented: technical student starting next month
  - Prioritisation of implementation order to be defined
- Encourage your local grid site to support the FCC VO (especially compute)


fccdirac.cern.ch