

III Tutorial on the ATLAS Offline Software

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

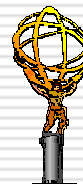
APPENDICES

Daniela Rebuzzi daniela.rebuzzi@pv.infn.it

Roma, 12 April 2005

Outline

- *Generators*
- *Simulation*
- *Digitization*
- *Batch running*



List of conventions

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- Any command you should type can be found in `~drebuzzzi/public/Tutorial/commands_list`, you can cut&paste from it
- Any specific word has its explanation in the [dictionary](#)
- Define the lines you have to type on your shell/prompt during this tutorial
- Explanation, hints
- ↔ Exercises which can be skipped in the case time is not enough



PART I: Generators

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

What you should expect to learn...

- How to run the generators in the ATLAS offline software environment (using the ASK interface)*
- How to generate events with PYTHIA*
- How to save generated events in a CBNT ntuple*
- How to save generated events in POOL persistency (to be re-read by the simulation, for instance)*
- How to run with the SingleParticle generator*
- How to run ATLFAST (ATLAS fast simulation) in ATHENA*
- Where to find generated events in the “Rome workshop” configuration and other useful links*



Event Generators

within the ATLAS offline software

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- The **Event Generators** model the physics processes that occur in high-energy collisions, like the ones at LHC
- This information is passed to the simulation which takes care of the trajectories of the particles and their interaction with the detector
- The **output** of a generator is a list of particles (incoming, created, decay products, etc.), their properties (particle PDGcode, vertex, four-vector) and their origin, just after the collision
 - The output format is the standard HepMC
 - In ATHENA, the output is written to a permanent store using POOL
- Each generator written in standalone version is made available in ATHENA through an appropriate interface package (`_i`)
 - This interface packages provide ATHENA-style algorithms which pick up the datacards for the generators via jobO, run the generators and allow for writing out the results in a common HepMC format utilizing POOL



Setup and run the Generators

□ Create a clean work directory

```
$> mkdir Tutorial
$> cd Tutorial
$Tutorial> mkdir Gen
$Tutorial> cd Gen
$Tutorial> ask
>>> runtime('8.8.1')
RunTime>
```

→ ASK= Athena Startup Kit

→ If you provide no arguments to `runtime()`, the latest available release will be selected

You are now ready to run the generators

□ As a test, run PYTHIA generator with the standard options (-b tells ATHENA to run in batch mode)

```
RunTime> athena.py -b Pythia_i/jobOptions.pythia.py>gen-log1
RunTime> theApp.run()
```

→ Even if not in your local directory, jobO are made available through the `JBOPTSEARCHPATH` variable

→ Note in the `gen-log1` output file the huge output from `DumpMC` at the end of the run, which is the ASCII representation of the result stored in `HepMC` format



A look at the PYTHIA jobO

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- From the ASK prompt, type the command to copy the `jobOptions.pythia.py`, it will be placed in your `~/Gen/run` dir

```
RunTime> getfiles("Pythia_i/jobOptions.pythia.py")
```

- Edit it by typing `vi run/jobOptions.pythia.py` (or your favorite editor) → ASK accept shell commands

```
File Edit Search Preferences Shell Macro Windows Help
#-----
# General Application Configuration options
#-----
theApp.setup( MONTECARLO )

include( "PartPropSvc/PartPropSvc.py" )

#-----
# Private Application Configuration options
#-----
theApp.Dlls += [ "TruthExamples", "Pythia_i" ]
theApp.TopAlg = ["Pythia","DumpMC"]
theApp.ExtSvc += ["AtRndmGenSvc"]
# Set output level threshold (2=DEBUG, 3=INFO, 4=WARNING, 5=ERROR, 6=FATAL )
MessageSvc = Service( "MessageSvc" )
MessageSvc.OutputLevel = 2
#-----
# Event related parameters
#-----
# Number of events to be processed (default is 10)
theApp.EvtMax = 5
#-----
# Algorithms Private Options
#-----
AtRndmGenSvc = Service( "AtRndmGenSvc" )
AtRndmGenSvc.Seeds = ["PYTHIA 4789899 989240512", "PYTHIA_INIT 820021 2347532"]
# AtRndmGenSvc.ReadFromFile = true;
Pythia = Algorithm( "Pythia" )
Pythia.PythiaCommand = ["pysubs msel 13","pysubs ckin 3 18.,"pypars mstp 43 2"]
#-----
```

- Needed for python setup
- jobO fragments which activates the PartPropSvc of ATHENA/Gaudi
- ATHENA loads the lib of the Algs
- Activate the ATHENA random Svc
- Increment this if you want less verbosity
- Number of events to be processed
- Set the random number seeds
- Generates Z+jets events, with p_T cut at 18 GeV and with γ and Z/ γ interference switched off



Modify the *PYTHIA* jobO

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- Change the “PythiaCommand” parameter to force $Z \rightarrow ee$ decay

```
#-----  
# inclusive Z production with the decay forced to e+e-  
#-----  
Pythia = Algorithm( 'Pythia' )  
Pythia.PythiaCommand = [  
  'pysubs msel 0',           → Allows the user to choose the decay mode  
  'pysubs msub 1 1',       → Create Z boson  
  'pypars mstp 43 2',     → switch off  $\gamma$  and  $Z/\gamma$  interference  
  'pydat3 mdme 174 1 0',  
  'pydat3 mdme 175 1 0',  
  'pydat3 mdme 176 1 0',  
  'pydat3 mdme 177 1 0',  
  'pydat3 mdme 178 1 0',  
  'pydat3 mdme 179 1 0',  
  'pydat3 mdme 182 1 1', → Switch on the  $Z \rightarrow ee$  decay mode, all the  
  'pydat3 mdme 183 1 0', other decays are switched off  
  'pydat3 mdme 184 1 0', → Switch for the  $Z \rightarrow \mu\mu$  decay (here off)  
  'pydat3 mdme 185 1 0',  
  'pydat3 mdme 186 1 0',  
  'pydat3 mdme 187 1 0' ]
```

- Run athena with the **local** jobOptions.pythia.py

```
RunTime> athena.py -b run/jobOptions.pythia.py
```



Exercise 1

Event generation with PYTHIA

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Edit the `jobOptions.pythia.py`
2. Modify it in order to generate a sample of 100 $Z \rightarrow \mu\mu$ events with a low p_T cut at 6 GeV
3. Run ATHENA with your local, modified jobOption

To check the output, you could add these two lines to the `PythiaCommand` in the `jobO`

```
'pyinit pylisti 12',
```

→ Dumps all the particle and decay data

```
'pyinit output junk.dat'
```

→ Causes all the PYTHIA output to dump into the file `junk.dat`



CBNT Output

- *The algorithm which fills the combined ntuples can take the generator output from the Transient Data Store and write it to an ntuple*
- *Add the following lines to the jobO file*

```
theApp.Dlls+= [ 'RootHistCnv' ] {  
theApp.HistogramPersistency = 'ROOT'  
NTupleSvc=Service( 'NTupleSvc' )  
NTupleSvc.Output=["FILE1 DATAFILE='cbnt.root ',OPT='NEW'"]  
include( 'CBNT_Athena/CBNT_Athena_jobOptions.py' ) }  
include( 'CBNT_Athena/CBNT_EventInfo_jobOptions.py' )  
include( 'RecExCommon/CBNT_Truth_jobOptions.py' )  
CBNT_Athena.NtupleLocID = '/FILE1/CBNT/t3333'
```

{ *Histos and ntuples will be saved using ROOT. The application manager is instructed to use it and the output ntuple name is specified*

} *Options and configurations to use the combined ntuples and to save both the run and event, and the Truth information, in them*

Conventional identifier (t3333) and location, in the output file, of the combined ntuple



Exercise 2

Checking the event information at the generation level

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. *Edit your local jobOptions.pythia.py*
2. *Copy at the end of it (after “Ntuple service output”) the lines for saving the generator output in a CBNT*
3. *Rerun ATHENA using the local, modified jobO*
4. *At the end of the run, there will be a “cbnt.root” file in the run directory*
5. *Open it with ROOT, locate the ntuple called t3333 and display few parameters (ex.GenEta) to verify that its contents make sense*



POOL Persistification

- *The technology needed for writing the output to a permanent store is delivered by the POOL project*
- *Applying the POOL services to persistify objects is transparent, but the job needs to be configured to use it, and it must be specified exactly which are the objects that need to be persistified*

```
# the following five lines are added to use POOL persistency
include('AthenaPoolCnvSvc/WriteAthenaPool_jobOptions.py' ){
include('GeneratorObjectsAthenaPool/GeneratorObjectsAthenaPool_jobopt
ions.py' )
Stream1.ItemList += [ '2101#*', '133273#*' ] }
include( 'AthenaSealSvc/AthenaSealSvc_joboptions.py' )
AthenaSealSvc.CheckDictionary = TRUE
Stream1.OutputFile = 'pythia.pool.root'
```

{ *load the general options needed to use POOL and the specific ones to persistify the generator object*

} *the class identifiers of the generator objects: all object in the Transient Data Store that corresponds to these class IDs will be selected for writing. The variable “Stream 1” is conventional and defined in the standard POOL write options*

optional: configuration for the in-memory verification of the dictionary

the name of the output from POOL. The extension is .root because ROOT I/O is used as default. The default location is the local disk in the dir where you execute ATHENA



Exercise 3

Writing events on disk with POOL after filtering them

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Copy in your `~/Gen/run` directory the `JobOptions.pythia.higgs.filter.py jobO` from my `~/drebuzzzi/public/Tutorial`
2. Edit the file and try to understand which type of events it produces
3. Check whether it has the lines to persistify the output file
 - If POOL is not configured in the jobO, the output is only available in-memory through the TDS
4. Run ATHENA with this jobO file and check with the DumpMC output whether the supposed events are actually produced
5. Check in your run directory that you have a root file (it is the POOL output file with HepMC formatted events)
 - Until 10.0.1 also a `PoolFileCatalog.xml` is created. The catalog file allowed POOL to locate files from the identifier which is given in the configuration.
6. Reread the events using the `readGenEventFromPool.py` in the `GeneratorObjectsAthenaPool` package
 - Retrieve this jobO in your local dir with “getfiles” and edit it to properly set the name of the file to be re-read



Exercise 3a

Generating an event sample to be simulated later

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. *Modify the `JobOptions.pythia.higgs.filter.py` jobO in order to select Z decay into two muons only*
2. *Generate a sample of 1000 $H \rightarrow ZZ \rightarrow 4\mu$ events and store the output in the “`Higgs_4mu.pythia.pool.root`” POOL file*

→ We will use it for the next simulation session!

→ Keep also the corresponding `PoolFileCatalog.xml`, even if, starting from ATHENA release 10.0.1 this is not needed anymore!



Creating ATLFAST ntuples

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- ATLFAST is the **fast simulation** code of the ATLAS detector → it simulates the detector response through several parameterization tables
 - Its input is generated events in HepMC format, its output a ntuple file
- To start you need to check out from the repository the `GeneratorOptionsRome`, because this package is not include in the ATHENA release (and the `getfiles` script does not work, therefore)

```
RunTime> checkout ("Generators/GeneratorOptionsRome")
```

```
RunTime> make → Although there is no code in this package, you need to  
make it, for ASK to properly understand it
```

```
RunTime> athena.py GeneratorOptionsRome/rome_RunGen_filter_  
AtlfastCBNT_pool.py → Run the package from ASK in the standard way
```

- This job runs the `ParticleGenerator` (creates e^+), filters the events, calls ATLFAST to run on the selected events and makes the CBNT ntuples. It also saves the generated events which passed the selections to POOL persistency



ParticleGenerator

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- The ParticleGenerator package runs within ATHENA and puts single primary particles into the HepMC TDS
- To use it, add these two lines in the cmt/requirements file of your run package (e.g. TestRelease)

```
use ParticleGenerator ParticleGenerator-* Generators
use McEventSelector McEventSelector-* Generators
```

and add this part to your jobO

```
File Edit Search Preferences Shell Macro Windows Help
#--- ParticleGenerator -----
# PDGCodes
# e=-11, e+=-11, mu=-13, mu+=-13, pi+=211, pi--211, pi0=111
# gamma=22, geantino=999
# Energy and momentum now in MeV!!!!
theApp.Dlls +=["ParticleGenerator"]
# If you want to change the random number seed for your
#ParticleGenerator, uncomment the three following lines
#and replace second number in the AtRndmGenSvc.Seeds
#command with the one you want.
theApp.ExtSvc += [ "AtRndmGenSvc" ]
AtRndmGenSvc= Service("AtRndmGenSvc")
AtRndmGenSvc.Seeds = [ "SINGLE 2040160768 443921183" ];
# If you want to read seeds from a file uncomment the next line
#AtRndmGenSvc.ReadFromFile = TRUE;
ParticleGenerator = Algorithm( "ParticleGenerator" )
ParticleGenerator.orders = [
  "pdgcode: constant 13",
  # "energy: constant 180000",
  "energy: normal 166800 8830",
  "vertX: constant -27500.0",
  "vertY: flat -10.0 15.0",
  "vertZ: constant 0.0",
  "t: constant -27500.0",
  # "momX: fixed 1",
  # "momY: fixed 0",
  # "momZ: fixed 0",
  "phi: constant 0.01",
  "theta: constant 1.560"
]
1
```

→ The ParticleGenerator library

→ The ATHENA Random Number Svc and the initialization seeds

→ The ParticleGenerator Algorithm

→ The ParticleGenerator array of strings. Each string contains an order for any different kinematic variable

Refer to the ParticleGenerator manual for the complete list of the kinematic variables and their settings



↔ Exercise 4 (advanced)

Running ATLFAST on the previously generated $H \rightarrow 4\mu$ events

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- Starting from `GeneratorOptionsRome/rome_RunGen_filter_AtlfastCBNT_pool.py`, write a `jobO` which runs ATLFAST on the POOL file generated in Exercise 3a
- Look at the ATLFAST results, and compare them with parameters at the generation level
- Keep the ATLFAST ntuple for a comparison with the full simulation results

→ We will arrive up to the digitization step today, but you already know from Stefano's tutorial how to run the reconstruction!



↔ Exercise 5

Event generation with HERWIG, TAUOLA/PHOTOS

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. *Run athena to produce SUSY events using HERWIG and TAOULS/PHOTOS to decay the taus and handle photon radiation of an event*

```
RunTime> getfiles("Herwig_i/jobOptions.herwig.py")  
RunTime> getfiles("Pythia_i/mean.isawig")  
RunTime> athena.py -b run/jobOptions.herwig.py
```



Rome Generated Events

And other useful links

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

□ **List of Rome samples**

<https://uimon.cern.ch/twiki/bin/view/Atlas/RomeListOfSamples>

□ **ATLAS Generators**

<http://www-theory.lbl.gov/%7Eianh/monte/Generators/>

□ **PYTHIA manual**

<http://rampex.ihep.su/manuals/pythia6206.pdf>

□ **ParticleGenerator manual**

<http://www-theory.lbl.gov/~ianh/monte/Generators/ParticleGenerator/ParticleGenerator.pdf>

□ **The Athena Startup Kit web page**

<http://wlav.home.cern.ch/wlav/athena/athask/>



PART II: Simulation

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

What you should expect to learn...

- Which are the main “simulation ingredients”*
- How to run the Simulation in the ATLAS offline software environment*
- How to switch on/off detectors (DetFlags) and build the simulation automatically in accordance*
- How to select particle/energy/magnetic field (with SimFlags)*
- Access to different physics lists/regions/cuts*
- How to save the hits on POOL persistency*
- How to re-read simulate events and how to histogram the hits*
- How to navigate the simulation with AtlasG4Eng.G4Eng*
- How to visualize the geometry*



Atlas Simulation at a glance

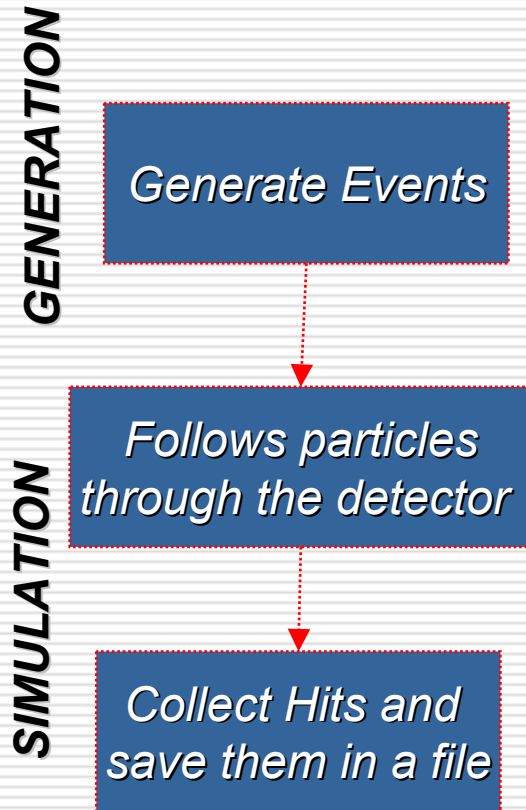
GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES



- The detector simulation takes as **input** the output from event generators (HepMC format events persisted with POOL), which consists of list of particles and their properties at specified vertices and follows the particles as they would traverse through the detectors
- Each time a particle hits a so-called “sensitive” detector element, a **hit** is recorded
- Hits are the **output** of the simulation, they are stored in HitCollections (AthenaHitsVector) and saved into a POOL file

→ Hits will be the input of the digitization, the simulation of the detector response



Hits and their Persistency

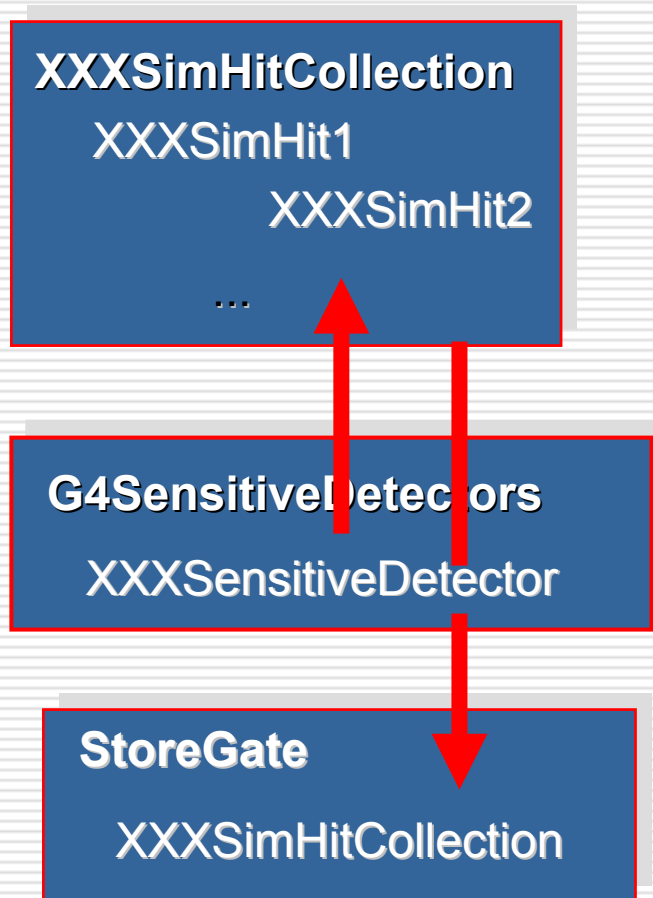
GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES



□ **HitCollections** are

- *separately built for each subsystem*
- *collected using AthenaHitsVector (sort of DataVector)*
- *persistified with POOL*
All you have to do is to set the hit output file name in the jobOption
- *can be read directly from SG via AthenaPool jobOptions*

- *Hits store information at tracking time, as (for instance) records of particles crossing a certain surface in the detector*
- *Completely transparent to the occasional user*



Simulation Ingredients

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

□ **Hit**

A snapshot/record of a physical interaction in a volume

□ **SensitiveDetector**

A class which utilizes geometrical/physical information to create hits. A SD object, when assigned to a volume, makes it sensitive

□ **PhysicsList**

A menu of physics processes, which will be activated in the detector simulation

□ **DetectorFacility**

A class which represents the entry point for a sub-detector geometry; sets up the geometry of the sub-detector envelope and its contents

□ **UserAction**

User defined class, utilized for interacting with the program at various levels (to add, for instance, analysis code)

□ **Geometry (external)**

Retrieved from GeoModel which is loaded as external service



A Simulation Application

- **Geometry Creation:** *the experimental layout is built by combining DetectorFacilities together*
- **Physics Initialization:** *a user can decide which physics menus will be run from a list of possible choices, or define his own*
- **Detector Sensitivity:** *selected volumes are made sensitive by assigning to them a SensitiveDetector object*
- **Magnetic Field:** *field maps can be defined for the whole detector/selected regions*
- **User Actions:** *users decide how to interact with the program, at the level of run/event/step/track/stack*
- **(Detector/event visualization)**



G4Atlas Simulation from 10.0.0

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

This is still in progress!

- Since ATHENA 10.0.0, the package AtlasG4Sim for the ATLAS G4 simulation is not maintained anymore
- A new Package hierarchy is built under Simulation/**G4Atlas** (without modifying any of the existing stuff)
 - Python simulation interface
 - G4macro commands are replaced by python, the steering through G4 macro files replaced by python scripts
 - Python commands are used for undertaking all necessary steps in initializing an application
- **G4AtlasApps** is a Python-coded package that can set up and run, within Athena, the Geant4 ATLAS full simulation (or any other ATLAS simulation like the one of the Combined Test Beam)
- G4AtlasApps interacts with FADS (Framework for ATLAS Detector Simulation) and with Geant4 through PyG4Atlas and PyLCGDict

→ The ATLAS 10.0.0 release is the last release in which the "G4 macro file structure" is kept updated, and is the test release for the new Python interface



Exercise 6

Run the G4Atlas simulation "out of the box"

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Create a directory for your simulation in your ~/Tutorial directory

```
$Tutorial> mkdir G4Sim  
$Tutorial> cd G4Sim
```

2. Set up the correct requirements file

```
$G4Sim> cp ~drebuzzzi/public/Tutorial/requirements .  
$G4Sim> source/afs/cern.ch/sw/contrib/CMT/v1r16p20040901/  
mgr/setup.sh  
$G4Sim> cmt config  
$G4Sim> source setup.sh -tag=10.0.2,opt  
$G4Sim> source /afs/cern.ch/atlas/software/dist/X.Y.Z/  
Simulation/G4Atlas/G4AtlasApps/G4AtlasApps/00-00*/cmt/  
setup.sh  
$G4Sim> get_files PDGTABLE.MeV
```

3. That is all! You can **run the ATLAS full simulation!**

```
$G4Sim> athena.py G4AtlasApps/jobOptions.G4Atlas_Sim.py
```

→ This test jobO uses the SingleParticle generator, simulate three events and writes out an output POOL file, "atlas_MyOutputFile.root", which contains the hits

→ You **do not need** any run dir and you do not need to checkout anything (only if you are using release 10.0.0 you have to check out and build Simulation/G4Utilities/G4PhysicsList)



Exercise 6a

Run the CTB simulation "out of the box"

GENERATORS

SIMULATION

DIGITIZATION

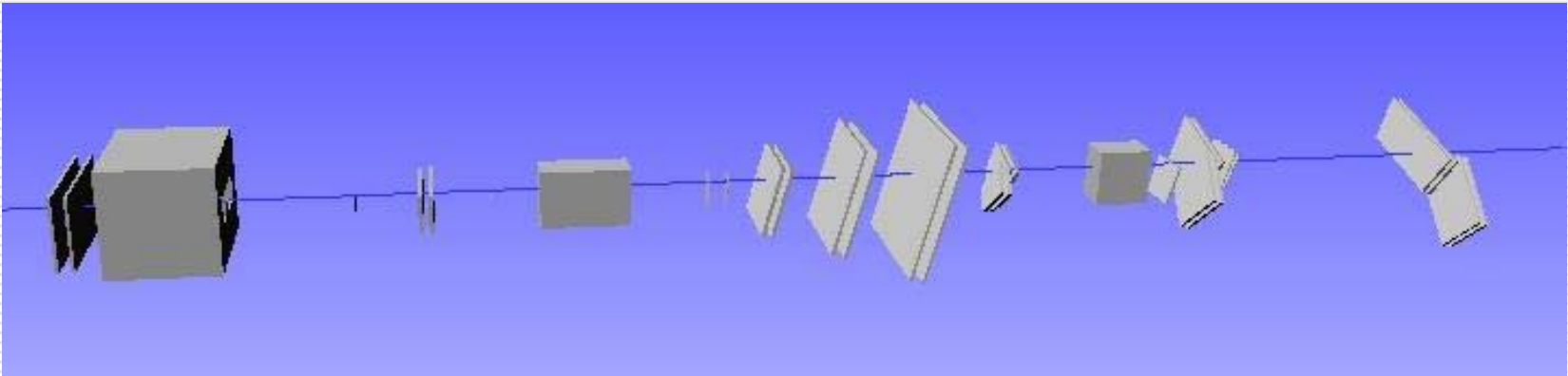
BATCH RUNNING

APPENDICES

1. From the previous environment you can also **run the CTB full simulation**

```
$G4Sim> athena.py G4AtlasApps/jobOptions.G4Ctb_Sim.py
```

→ Running ATLAS simulation or the CTB one is just a matter of jobO



→ But here we are not going to treat the CTB simulation....



Job Configuration

□ Retrieve the `jobOptions.G4Atlas_Sim.py` from the release

```
$G4Sim> get_files -jo G4AtlasApps/jobOptions.G4Atlas_Sim.py
```

→ Check that it is actually retrieved from the distribution

It is a **python script** which defines the configuration for the simulation

1. Job configuration using flags

At this level it is possible to decide, for instance, which detectors will be involved in the simulation, the particle type and energy, the persistency hit file, the geometry layout, the seeds for the generator, and many other options

For most of the use cases, the flag mechanism gives enough freedom to configure the job and allows to load the minimal number of libraries and dependencies

2. Job configuration using the Geant4 Engine (beyond the aim of this tutorial)

AtlasG4Eng.G4Eng is a python simulation engine which provides the possibility to interact with, navigate and fully configure the simulation



Detector Flags

- The Detector Flags describe the detectors involved in a simulation job

→ The “DetFlags” are in use also in other ATLAS software domains

(e.g. the reconstruction)

- The python line

```
from AthenaCommon.DetFlags import DetFlags
```

in the jobO, gives access to these flags, to their boolean values (True/False) and to the methods to set them

- After the previous Python "import statement" you may find, in the jobO, the following lines:

```
# - Select detectors
DetFlags.ID_setOn()
DetFlags.Calo_setOff()
DetFlags.Muon_setOff()
#
DetFlags.simulate.Truth_setOn()
```

which establish that for this example the simulation job will use only the Inner detector and will record the truth information



Simulation Flags

- *The Simulation Flags are used to configure specifically the simulation*
 - *They may carry also a value apart from the boolean status (True/False).*

- *They appear at the beginning of the jobO file and they give the possibility to set*

Energy GeneratorPath KinematicsMode ParticlePDG PersistencyDigit PersistencyHit PhysiscList Seeds SimLayout

- *Their description, statusOn/Off, actual value, default value, possible values and methods are described on the Web (atlas specific, ctb specific) and are always accessible from the athena prompt*

- *In the jobOption file or at the Athena prompt, the Python line:*

```
from G4AtlasApps.SimFlags import SimFlags
```

is enough to give full access to them

- *The SimFlags import method present in the jobO*

```
SimFlags.import_Flags('atlas_flags')
```

enrich the initial set of flags with all the flags (atlas specific, in this case) contained in the atlas_flags.py file (located in G4AtlasApps/python)



Flag Documentation

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

→ From the cvs repository under ~/doc directory of each G4Atlas package

The screenshot shows a web browser window displaying the documentation for G4AtlasApps. The main content area is titled "G4AtlasApps.atlas_flags (version 1.1)" and is located at the URL [/afs.cern.ch/user/g/gallasm/AtlasSw/base1001-d/InstallArea/python/G4AtlasApps/atlas](http://afs.cern.ch/user/g/gallasm/AtlasSw/base1001-d/InstallArea/python/G4AtlasApps/atlas). The page content is as follows:

Simulation specific flags.

Apart from the AthenaCommon.DetFlags and AthenaCommon.GlobalFlags the simulation jobs will need specific flags that are collected here as "SimFlags".

Package Contents

AtlasG4Eng	atlas_flags
PyG4Atlas	atlas_idet
SimFlags	atlas_materials
atlas_calor	atlas_mctruth
atlas_common	atlas_muon

Data

```
__all__ = ['PyG4Atlas', 'AtlasG4Eng', 'SimFlags', 'atlas_common', 'atlas_utilities', 'ctb_field', 'ctb_common', 'ctb_idet', 'ctb_calor', 'ctb_muon']
__author__ = 'A. Dell'Acqua, M. Gallas'
__description__ = 'Python interface for ATLAS Geant4 simulation.'
__version__ = '$Revision: 1.10 $'
```

Author

A. Dell'Acqua, M. Gallas

Classes

G4AtlasApps.SimFlags.flags(builtin .object)

- [Energy](#)
- [GeneratorPath](#)
- [ParticlePDG](#)
- [Seeds](#)
- [SimLayout](#)

class Energy(G4AtlasApps.SimFlags.flags)

[Energy](#) of the particle for the single particle generator.

The default value is 50000 MeV, and it must be integer/float.
The units are MeV

Method resolution order:

- [Energy](#)
- [G4AtlasApps.SimFlags.flags](#)
- [_builtin_.object](#)

Data and other attributes defined here:

```
StatusOn = True
```



A look at the Simulation job0

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

```
File Edit Search Preferences Shell Macro Windows Help
#-----
#
# Job options file for Geant4 Simulations
#
# Atlas simulation
#
#-----
#--- Detector flags -----
from AthenaCommon.DetFlags import DetFlags
# - Select detectors
DetFlags.ID_setOn()
DetFlags.Calo_setOff()
DetFlags.Muon_setOff()
#
DetFlags.simulate.Truth_setOn()

#--- Simulation flags -----
from G4AtlasApps.SimFlags import SimFlags
SimFlags.import_Flags('atlas_flags')

SimFlags.PersistencyHit.set_Value('atlas_MyOutputFile.root')

# - uses single particle generator
SimFlags.KinematicsMode.set_Value('SingleParticle')
SimFlags.ParticlePDG.set_Value('11')
# - reads events already generated
#SimFlags.KinematicsMode.set_Value('ReadGeneratedEvents')
# - uses a given generator
#SimFlags.KinematicsMode.set_Value('EventGenerator')
#SimFlags.GeneratorPath.set_Value('G4AtlasApps/Z-mumuOptions.py')
#SimFlags.GeneratorPath.set_Value('GeneratorOptionsRome/rome.004201.JimmyZee.py')

#--- Event related parameters -----
# Number of events to be processed (default is 10)
theApp.EvtMax = 3

#--- Output printout level -----
#output threshold (2=DEBUG, 3=INFO, 4=WARNING, 5=ERROR, 6=FATAL)
#you can override this for individual modules if necessary
MessageSvc = Service( "MessageSvc" )
MessageSvc.OutputLevel = 4
```

→ **Import of the Detector flags**
(detectors on/off, MCTruth)

→ **Import of the Simulation flags**
→ **With this SimFlags method, the initial set of flags contained in atlas_flags.py (python script which contains common flags to be selected when running the atlas simulation) are imported**

{ **SimFlags used here to Set the KinematicMode to SingleParticle → the ParticleGenerator is used**

} **Number of events to be processed**
Output printout level



A look at the simulation job0 (cont'd)

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

```
File Edit Search Preferences Shell Macro Windows Help
#-----
# Job configuration
# ***>> Do not add flags or simulation options below this line
#-----
#--- Event Kinematics ---
if not(theApp.properties().get('ExtSvc').__contains__('AtRndmGenSvc')):
# only in case the generator was not used before we will run
# this part
    if SimFlags.KinematicsMode.Value=='SingleParticle':
        include ("G4AtlasApps/SingleParticleOptions.py")
    elif SimFlags.KinematicsMode.Value=='EventGenerator':
        include (SimFlags.GeneratorPath.Value)
    elif SimFlags.KinematicsMode.Value=='ReadGeneratedEvents':
        include ("G4AtlasApps/GeneratedEventsOptions.py")

#--- GeoModel stuff ---
include ("G4AtlasApps/AtlasGeoModelOptions.py")

theApp.setup( MONTECARLO )
#--- Atlas setup description ---
include("G4AtlasApps/AtlasSim.py")
#G4Command=AtlasG4Eng.G4Eng.gbl.G4Commands()
#G4Command.tracking.verbose(1)
AtlasG4Eng.G4Eng.init_Simulation(3)

#--- End jobOptions.G4Atlas_Sim.py file ---
```

→ Job configuration part. All the relevant information for the simulation should go **before** this line labeled with "Job configuration", below which the configuration starts

→ setup the GeoModel service and options

```
File Edit Search Preferences Shell Macro Windows Help
# This is the GeoModel initialization snippet. There is no reason
# for changing this bit, unless one wants to run a different
# detector configuration

#--- GeoModel stuff ---
theApp.Dlls += [ "Geo2G4" ]
theApp.ExtSvc += [ "Geo2G4Svc" ]
DetDescrVersion = "Rome-Initial"
include ("AtlasGeoModel/SetGeometryVersion.py")
include ("AtlasGeoModel/GeoModelInit.py")
GeoModelSvc = Service( "GeoModelSvc" )
```

{ The entry point for the simulation engine, main configuration for the ATLAS simulation application: set the Hit Persistency, setup G4Svc and G4AtlasAlgs, initialize the application, set the extra materials, creates the ATLAS volumes, build the subdetectors, select the physics list

} Bring the simulation to a different init state (1 to 3) → 3 means "ready to run"

G4Eng instance of the simulation engine here is used to set the init level of the simulation



Running in interactive mode

- *The Athena interactive prompt can be accessed with the following command*

```
$G4Sim> athena -i jobOptions.G4Atlas_Sim.py
athena>
athena> theApp.run()
athena> theApp.finalize()
```

→ *Starts the simulation application*

→ *Finalize it*

→ *apart from the possibility to perform some shell commands (like: ls, pwd, cp), the athena prompt offers access to the configuration of simulation job and to the help needed to know about the simulation options & flags*

From the athena prompt there are three important objects which can be accessed

```
athena> AtlasG4Eng.G4Eng.Name
athena> DetFlags.Print()
athena> SimFlags.print_Flags()
```

→ *The G4Engine, here its name is retrieved*

→ *The DetFlags*

→ *The simulation options and SimFlags*

All DetFlags and SimFlags can be navigated from the athena prompt typing their name followed by . and pressing the <Tab>

→ *Flags are auto-documented*



↔ Exercise 7

Navigating the flags and getting the help

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

Run athena in interactive mode with your local jobO

- 1. Check the status of the Detector and Simulation Flags*
- 2. Navigate the Flag classes, asking for help*

```
athena> help(SimFlags.ParticlePDG)
```

- 3. access flag properties, by typing*

```
athena> SimFlags.NameOfFlag.Value
```

- 4. use flag methods*

```
athena> SimFlags.NameOfFlag.print_FlagFull()
```

- 5. Check in particular*

SimFlags.Energy	SimFlags.GeneratorPath
SimFlags.KinematicsMode	SimFlags.ParticlePDG
SimFlags.PersistencyHit	SimFlags.SimLayout
SimFlags.PhysiscList	SimFlags.Seeds



SimFlags.PhysicsList

How to change the physics and to change the cuts

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- ❑ *Be careful with the Physics Lists, you can also build your own, but you are recommended to choose one from the menu*

```
athena> SimFlags.PhysicsList.print_FlagFull()  
G4AtlasApps.SimFlags:: PhysicsList True QGSP_GN  
['ExN01', 'ExN02', 'ExN03', 'ExN04', 'Fast_Physics',  
'LHEP_BERT', 'LHEP_GN', 'QSGP_BERT', 'QGSP_GN'] ['str']
```

Refer to G4 doc for details of the various Physics Lists

- ❑ *When you are happy with a choice (say, Fast_Physics), write the following lines in the jobO, before setting the init level*

```
myphysics=AtlasG4Eng.G4Eng.Dict.get('physics')  
myphysics.Name='Fast_Physics'
```

- ❑ *To change the cuts in range on particles, write just below*

```
myphysics.Value_gen_cut=5*mm
```

- ❑ *In the immediate future this will be done in the SimFlags part with*
`SimFlags.PhysicsList.set_Value('Fast_Physics')`



↔ Exercise 8

Running with different physics conditions

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Edit the `jobOptions.G4Atlas_Sim.py`
2. Use the `SimFlags` to select the `Fast_Physics` list and run 10 events
→ only tracking is performed on particles, all the physics processes switched off
3. Edit again the `jobOptions.G4Atlas_Sim.py` and set the simulation init level to 1
4. Run `athena` interactively and from the prompt, use the `SimFlags` to raise the cut in range on particles to 50cm
→ `dE/dx` only, a la `Geant3`
5. Set the init level to 3 and run 10 events
→ The simulation can be initialized at different level
 - `Init_level = 0` → not initialized
 - `Init_level = 1` → Init Physics
 - `Init_level = 2` → Init G4
 - `Init_level = 3` → Init the Physics Regions



SimFlags.KinematicsMode

How generators can be plugged in

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- G4Atlas expects an HepMC event on input. You have three possibility

```
athena> SimFlags.KinematicsMode.print_FlagFull()  
G4AtlasApps.SimFlags:: KinematicsMode True SingleParticle  
{ ['SingleParticle',] 'EventGenerator', 'ReadGeneratedEvents'  
 ['str']
```

- { ATLAS SingleParticle generator

- } G4 specific event generators (like Pythia or Herwig)

Event file read back from POOL files

SimFlag.KinematicsMode allows to choose the generator

- According to the chosen generator, one needs to configure (via SimFlags) the corresponding flags

like the **PDG code** (SimFlags.ParticlePDG) in case of the 'SingleParticle' or the **generator path** (SimFlags.GeneratorPath) in case of the 'EventGenerator'

- The recommended way to run the simulation consists in **leaving the jobO as it is and prepending** to it the jobOption for the event generation

```
$G4Sim> athena.py GeneratorOptionsRome/rome.004204.Jimmy  
Wmunu.py G4AtlasApps/jobOptions.G4Atlas_Sim.py
```



Exercise 9

KinematicsMode='SingleParticle'

1. Edit the `jobOptions.G4Atlas_Sim.py` and modify it to run

- only with the Muon System
- generating 1000 μ^- (PDGcode=13) at 1000 GeV using ParticleGenerator
- (with the Fast_Physics list)

→ Once the jobO is local, remember to run it with

```
$G4Sim> athena.py jobOptions.G4Atlas_Sim.py
```

2. Save the hits in the POOL file “`muminus.pT1000.pool.root`”



MDTDigitValidation

To check the MDT hits position

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- The `MuonDigitization/MuonDigiExample/MDTDigitValidationOptions.py` runs the validation algorithm for the MDT digits basically, but it contains information on the MDT hits as well (which are used as comparison to validate the digitization)
- It runs on a hit file, performs the digitization on the fly and histograms hits and digits parameters + the Truth at the entrance of the Muon System
- Run it with

```
$G4Sim> athena.py MDTDigitValidationOptions.py
```

It produces a ROOT file, “`mdtval.root`”, which you can open with ROOT

- The parameters *related to the hits* are the following
 - Position of the hits in the global ATLAS reference system
`hit_gx_wire, hit_gy_wire, hit_gz_wire`
 - *Eta and phi of the hits*
`hit_phi_wire, hit_eta_wire`
 - *Eta and phi of the generated track at vertex (HepMc::GenVertex)*
`gen_phi, gen_eta`
- *P, pT and PDG code of the track at MSEL (McTruth/TrackRecord)*
`mse1_p, mse1_pt, mse1_pdg`



Exercise 9a

Reading hits with `MDTDigitValidationOptions.py`

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Check out the `MuonSpectrometer/MuonDigitization/MuonDigiExample` package and compile it
 2. Copy the `MDTDigitValidationOptions.py` in your run dir, from the repository or from my `~drebuzzi/public/Tutorial`
 - Edit it to set the correct input file name → “`muminus.pT1000.pool.root`”
 - Run athena to create the `mdtval.root` nutple
 2. Produce a scatter plot η - ϕ to check the geometry acceptance in the `MuonSystem`
 - For the hits
 - For the generated tracks
 3. compare the muon momentum at vertex with the muon momentum at the entrance of the `MuonSystem`
 - You can have an idea of the energy loss in the calorimeters
- The air volume MSEL (Muon System Entry Layer) allows to check the track parameters at the entrance of the Muon System. The associated Sensitive Detector is defined in `Simulation/G4Sim/MCTruth/TrackRecorderSD` class



Exercise 10

KinematicsMode='EventGenerator'

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Edit the `jobOptions.G4Atlas_Sim.py` and check that the `DetFlags` for the Muon System and for the calorimeters are set to `On()`
2. Copy the `GeneratorOptionsRome/rome.004202.JimmyZmumu.py` jobO in your work dir
3. Edit it and comment out the part of the POOL persistency
 - Otherwise, what happens is that the POOL generator objects will overwrite the simulation ones and no output will be saved from your simulation job
4. Run the simulation prepending `rome.004202.JimmyZmumu.py` jobO, to generate 10 $Z \rightarrow \mu\mu$ events and simulate them “on the fly”
5. Save the hits in the file “`Zmumu_Jimmy.pool.root`”



ReadMuonSimHits algorithm

To dump the MuonHits

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- This monitoring algorithm, which is located in the repository under `MuonSpectrometer/MuonDigitization/MuonDigiExample` can be used to check whether a root hit file contains Muon hits at the MCTruth at the entrance of the MuonSpectrometer
- This algorithm will simply dump hits out if not-empty Muon XXXSimHitCollections are found on StoreGate

→ But sometimes this is enough to know whether you did well in the simulation

- To use it, copy the jobO from `MuonDigiExample/share` or from my `~drebuzzi/public/Tutorial`, and edit it to set the correct input file name
- Run it with

```
$G4Sim> athena.py ReadMuonSimHitsOptions.py
```



Exercise 10a

ReadMuonSimHitsOptions.py

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Check with `ReadMuonSimHitsOptions.py` that the simulation run in the previous exercise correctly produced hits in the Muon System
 - Edit `ReadMuonSimHitsOptions.py` and set the input file name to “`Zmumu_Jimmy.pool.root`”
 - Run athena with `ReadMuonSimHitsOptions.py`



Exercise 11

KinematicsMode='ReadGeneratedEvents'

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Read the $H \rightarrow ZZ \rightarrow 4\mu$ event file created in the Generators part of the tutorial and simulate it

- In the `jobOptions.G4Atlas_Sim.py`, set the KinematicMode to 'ReadGeneratedEvents'
- Retrieve `GeneratedEventOptions.py` with `get_files`, open it and set the full path of the "Higgs_4mu.pythia.pool.root" file
- Run athena with

```
$G4Sim> athena.py GeneratedEventOptions.py  
jobOptions.G4Atlas_Sim.py
```

2. Run athena with `ReadMuonSimHitsOptions.py` on the output of the simulation to check whether hits in the Muon System have been correctly produced



Inspecting the Simulation

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- *PyG4Atlas defines the python classes for: DetectorFacilities, Sensitive Detectors, Mctruth strategies, PhysicsRegions and Cuts, PhysicsLists, UserActions...*
- *The PyG4Atlas.G4AtlasEngine puts every thing together (it should be possible to access any python object involved) and takes care of the different phases of initialization, log-service, etc...*
- *The AtlasG4Eng.G4Eng is the python simulation engine which allows to directly interact with FADS and G4*
- *It can be accessed from the athena prompt. Typing*

```
athena> AtlasG4Eng.G4Eng. <Tab>
```

List the properties and the methods for the simulation

It gives access to the G4 commands, for instance

- *All G4 command are directly available via the G4Commands interface (from the athena prompt or in the jobO)*

```
athena> G4command=AtlasG4Eng.G4Eng.gbl.G4Commands ()
```

```
athena> G4command.tracking.verbose(1)
```

→ *This command set the tracking to a medium verbosity level (default=0)*



Navigating the Simulation

How to visualize the detector

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- *From the shell prompt, set the following environment variables*

```
$> export G4VRMLVIEW_VIEWER=/afs/cern-ch/user/d/drebuzzi/  
public/Tutorial/vrmlview  
$> export LD_LIBRARY_PATH=/afs/cern-ch/user/d/drebuzzi/  
public/Tutorial/$LD_LIBRARY_PATH
```

- Check if everything is OK by typing*

```
~drebuzzi/public/Tutorial/vrmlview
```

a vrmlview window should open

- *OK, close it and run athena in interactive mode*

```
athena> simControl=AtlasG4Eng.G4Eng.gbl.SimControl()  
athena> simControl.initializeGraphics()  
athena> G4command=AtlasG4Eng.G4Eng.gbl.G4Commands()  
athena> G4command.vis.open("VRML1FILE")  
athena> G4command.vis.drawVolume()  
athena> G4command.vis.viewer.flush()
```

→ be careful because if it could take a lot of time!
Check which volumes are set to visible, first



Navigating the Simulation

Colors, visibility and elements

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- *Python is used to define colors (share/color.py) and elements (in AtlasG4Eng.py)*

```
Color("AliceBlue", 240, 248, 255)
Element("Hydrogen", "H", 1, 1.00797*g/mole, 0.0708*g/cm3)
```

- *Volume visibility attributes can be checked and changed interactively in G4Atlas from the athena prompt, using the AtlasG4Eng.G4Eng*

```
athena> simControl=AtlasG4Eng.G4Eng.gbl.SimControl()
athena> geoMenu=simControl.geomMenu()
athena> geoMenu.SetInvisible("*")
athena> geoMenu.SetVisible("Pixel::Pixel")
athena> geoMenu.SetColor("Pixel::SensitiveWafer",
    "AliceBlue")
```



Exercise 12

Visualizing the detector

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. *Run athena in interactive mode*
2. *Set any volume to invisible*
3. *Set to visible only the detector envelopes*
4. *Visualize the detector envelopes*
5. *Run the geantino map to check detector positions*
 - *Edit the jobOptions.G4Atlas_Sim.py*
 - *Select the ParticleGenerator and set “999” as PDGcode for the geantinos*
 - *Uncomment these two lines in the jobO*

```
G4Command=AtlasG4Eng.G4Eng.gbl.G4Commands ()  
G4Command=tracking.verbose(1)
```
 - *Run athena with jobOptions.G4Atlas_Sim.py saving the output the file “geant_map.out”*



Documentation and useful links

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

□ **G4 Simulation web page**

<http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/simulation/geant4/>

□ **Status of Simulation production**

<http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/testbeam/simulationCTB/productionMC.html>

□ **Geant4 ATLAS Application**

http://atlas-sw.cern.ch/cgi-bin/viewcvs_atlas.cgi/offline/Simulation/G4Atlas/G4AtlasApps/doc/index.html

□ **CVS repository**

<http://atlas-sw.cern.ch/cgi-bin/viewcvs-atlas.cgi/>



PART III: Digitization

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

What you should expect to learn...

- How to run the Atlas Digitization in the Atlas offline software environment*
- How to customize the Atlas Digitization*
- How the infrastructure to run the pileup works*
- Basics on the Muon Digitization*
- How to customize the MDT_Digitization*
 - *Changing digitization tool*
 - *Digitizing with customized rt relation*



Atlas Digitization

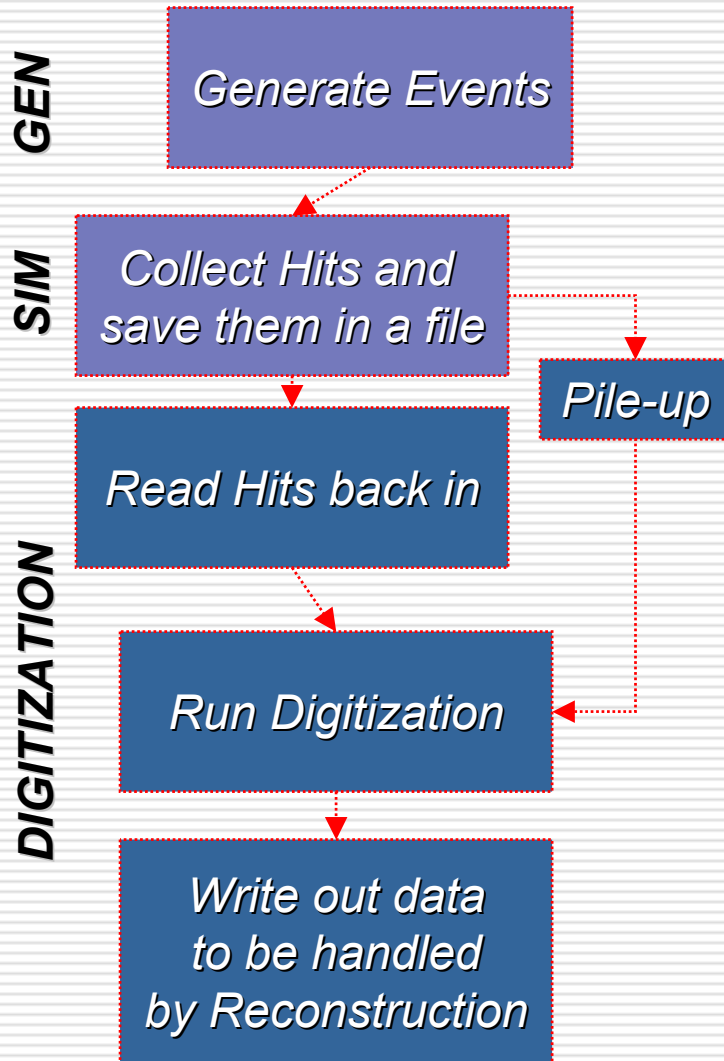
GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

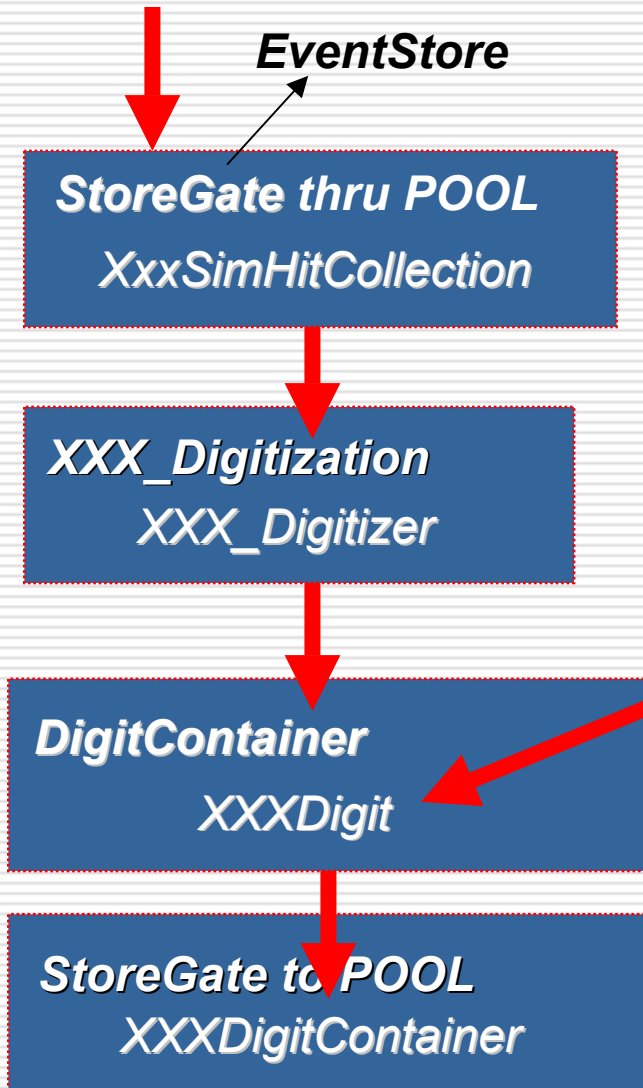


- The **digitization** takes as **input** the output from the simulation (which consists of HitCollections) and applies the effect resulting from the detector and its electronics
- The **output** are the so-called digits or the RDOs (Raw Data Objects = object copy of the bytestream content) which are the digitization persistified objects
- Digits or RDOs are the final product of the simulation chain and are fed into the reconstruction

→ any reference to the simulation information is lost after the digitization, digits and RDOs do not carry any link to the original track (a part from MuonSimData)



Atlas Digitization



SimHitId from simulation unfolded to get hit geometrical information

Each **XxxDigit** has an **offline identifier (OID)** which is the connection to the reconstruction

→ Separation between event and detector data → connection via identifier objects

IdHelpers
XxxIdHelper

OIDs initialized by the **IdHelpers** from the ID dictionaries

initialized from **GeoModel**

Output of digitization: **Digits or RDOs**

- persistified with **POOL**
- defined by the reconstruction group
- resemble the detector output (same objects as for real data)



Exercise 13

Running the ATLAS Digitization "out of the box"

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Create a clean directory to run the digitization, under your ~/Tutorial directory

```
mkdir Dig
cd Dig
get_files PDGTABLE.MeV
```

2. Run the ATLAS Digitization with ASK

```
$> ask
>>> runtime('10.0.1')
RunTime> athena Digitization/DigitizationConfig.py
Digitization/AtlasDigitization.py
```

→ This default jobO runs on a DC2 hit file, writing out RDOs

→ As for the simulation, you do not need to check out any pseudo-package



A look at the Digitization jobO

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

The AtlasDigitization.py jobOptions reads the POOL file generated by the simulation and writes out another POOL file with the digit objects

```
File Edit Search Preferences Shell Macro Windows
#####
#
# Job options file to run:
#   Digitization
#   LVL1 Simulation
#   ByteStream conversion
#
# Author: Davide Costanzo
#-----
#-----
# Get Digitization Flags (This sets GLocal and Det Flags)
#-----
# include ( "Digitization/Digitization_flags.py" )
# doFileUp set as or of doMinimumBias and doCavern
if "doFileUp" in dir() :
    raise "Please do not set doFileUp, use doMinimumBias and/or doCavern ins
else :
    doFileUp = doMinimumBias or doCavern

#switch off tasks
if not doFileUp:
    DetFlags.pileup.all_setOff()

# If write digits then switch off RDO making for the muons
# to work with release 9.0.2
if writeMuonDigit:
    DetFlags.writeRDOPool.Muon_setOff()
    DetFlags.LVL1_setOff()

if doFileUp and not (theApp.EventLoop=="FileUpEventLoopMgr") :
    raise "Please use the flag -p Digitization/pileUpBootstrap.py"
```

{ This DigitizationFlags include allows to import the flags which configure the digitization job

```
File Edit Search Preferences Shell Macro Windows Help
#
# Digitization_flags.py. Set the flags needed by digitization
# this is a readaptation of David's RecExCommon_flags.py
#
# some very general flags, EvtMax, SkipEvents, PoolESDInput etc...
include ("AthenaCommon/AthenaCommonFlags.py")
# prevent this file to be included a second time
include.block("AthenaCommon/AthenaCommonFlags.py")
# prevent the current file to be included a second time
include.block("Digitization/Digitization_flags.py")
#guard to include it only once
# use python ntuple to store variables (from wim)
# these are default values,
DigitizationFlags = {
    'doCaloNoise' : True,
    'doMinimumBias' : False,
    'doCavern' : False,
    'numberOfCollisions' : 23,
    'initialBunchCrossing' : -30,
    'finalBunchCrossing' : 4,
    'writeMuonDigit' : False,
    'minBiasInputCols' : ["/afs/cern.ch/atlas/offline/data/
"/afs/cern.ch/atlas/offline/data/
'cavernInputCols' : ["/rfio/castor/cern.ch/atlas/proje
}
# set variable to default if not already set
for o in [ o for o in DigitizationFlags.keys() if not o in varInit
exec '%s = DigitizationFlags[ "%s" ]' % (o,o)
#-----
# Set Global flags for this run
#-----
if 'GlobalFlags' in varInit:
    print "GlobalFlags already imported:"
else:
    from AthenaCommon.GlobalFlags import GlobalFlags
    GlobalFlags.DetGeo.set_atlas()
    GlobalFlags.DataSource.set_geant4()
    GlobalFlags.InputFormat.set_pool()
#-----
# Set Detector flags for this run
#-----
from AthenaCommon.DetFlags import DetFlags
#select detectors
DetFlags.ID_setOn()
DetFlags.Calo_setOn()
DetFlags.Muon_setOn()
DetFlags.Truth_setOn()
DetFlags.LVL1_setOn()
DetFlags.simulate.all_setOff()
DetFlags.makeRIO.all_setOff()
DetFlags.writeBS.all_setOff()
DetFlags.readRDOBS.all_setOff()
DetFlags.readRIOBS.all_setOff()
DetFlags.readRIOPool.all_setOff()
DetFlags.writeRIOPool.all_setOff()
# If write digits then switch off RDO making for the muons
# to work with release 9.0.2
if writeMuonDigit:
    DetFlags.writeRDOPool.Muon_setOff()
    DetFlags.LVL1_setOff()
```

→ Import and set of DetFlags



A look at the Digitization jobO

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

{ print DetFlags, GlobalFlags and DigitizationFlags

File Edit Search Preferences Shell Macro Windows

```
#print job configuration
DetFlags.Print()
GlobalFlags.Print()
# dump flag content
print "Digitization_flags.py flags values:"
for o in DigitizationFlags.keys():
    exec 'print "%s =" ,%s ' % (o,o)

include("Digitization/ConfigDigitization.py")

# Pool input (Change this to use a different file)
EventSelector = Service( "EventSelector" )
EventSelector.InputCollections = PoolHitsInput }

if doMinimumBias:
    minBiasEventSelector = Service( "minBiasEventSelector" )
    minBiasEventSelector.InputCollections = minBiasInputCols
if doCavern:
    cavernEventSelector = Service( "cavernEventSelector" )
    cavernEventSelector.InputCollections = cavernInputCols

# Pool Output (Change this to use a different file)
if DetFlags.writeRDOPool.any_on() or writeMuonDigits:
    Stream1.OutputFile = PoolRDOOutput

#
#-----
# Set output level threshold (2=DEBUG, 3=INFO, 4=WARNING, 5=ERROR)
#-----
MessageSvc = Service( "MessageSvc" )
MessageSvc.OutputLevel = 3
#-----
# Event related parameters
#-----
# Number of events to be processed (default is 10)
theApp.EvtMax = EvtMax
# Number of input events to be skipped
EventSelector.SkipEvents = SkipEvents
```

} reading from the hit file to be set in the Digitization Config.py, prepended to this jobO when running

File Edit Search Preferences Shell Macro Windows Help

```
#check job configuration
if not(DetFlags.geometry.any_on()):
    AthError( "this digitization job needs some DetFlags.geometry On" )
if not(DetFlags.digitize.any_on()):
    AthError( "this *digitization* job needs some DetFlags.digitize On" )
if not(DetFlags.writeRDOPool.any_on()):
    log.warning( "this digitization job will not write any RDO object" )

#Pool input
include( "AthenaPoolCnvSvc/ReadAthenaPool_jobOptions.py" )
# EventInfo Converters
include( "EventAthenaPool/EventAthenaPool_joboptions.py" )

# GeoModel stuff:
#-----
include( "AtlasGeoModel/SetGeometryVersion.py" )
include( "AtlasGeoModel/GeoModelInit.py" )

if DetFlags.pileup.any_on():
    include( "Digitization/ConfigPileUpEventLoopMgr.py" )

# in any case we need the PileUpMergeSvc for the digitize algos
theApp.Dlls += [ "PileUpTools" ]
theApp.ExtSvc += [ "PileUpMergeSvc" ]
```

→ POOL persistification

→ GeoModel settings

```
#subdetector-specific configuration
include( "Digitization/DetectorDigitization.py" )

if DetFlags.writeRDC
```

File Edit Search Preferences Shell Macro Windows Help

```
# Pool Output
theApp.OutStream
theApp.OutStream
Stream1.EvtConve
Stream1.ItemList
Stream1.ForceRes

# Author: Davide Costanzo
# Contacts:
# Inner Detector: Davide Costanzo
# LArCalorimeter: Guillaume Unal
# TileCalorimeter: Sasha Solodkov
# MuonSpectrometer: Daniela Rebutti, Retevi Assamagan
# LVL1 Simulation: Tadashi Maeno
# ByteStream: Hong Ma

# Inner Detector
if DetFlags.ID_on():
    include( "Digitization/InDetDigitization.py" )

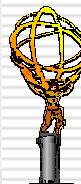
# Calorimeters
if DetFlags.Calo_on():
    include( "Digitization/CaloDigitization.py" )

# Muon Digitization
if DetFlags.Muon_on():
    include( "Digitization/MuonDigitization.py" )

# MC Truth info
if DetFlags.Truth_on():
    include( "Digitization/TruthDigitization.py" )

#lvl1 trigger simulation
if DetFlags.digitize.LVL1_on():
    include( "Digitization/LVL1Digitization.py" )
```

→ The subdetectors provide digitization jobO fragments



Customize the Digitization

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- The digitization needs the geometry

→ it uses the methods of the so-called GeoModel readout geometry to retrieve information about geometry parameters needed by the digitization procedures

→ **always** select the same geometry version as used to simulate events

- To digitize the events simulated in the previous section of this tutorial, you need to choose the “Rome workshop” configuration

To select it, edit the file *DigitizationConfig.py* and change the line

```
DetDescrVersion='DC2'
```

to

```
DetDescrVersion='Rome-Initial'
```

- If you want to write out MuonDigits, add these lines in the *AtlasDigitization.py* jobO

```
#-----  
# Write Muon digits instead of RDOs  
#-----  
writeMuonDigit = True  
from AthenaCommon.DetFlags import DetFlags  
DetFlags.writeRDOPool.Muon setOff()
```



ReadMuonDigits algorithm

To dump the MuonDigits

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- *This monitoring algorithm, which is located in the repository under MuonSpectrometer/MuonDigitization/MuonDigiExample can be used to check whether a root hit file contains MuonDigits*
- *It will simply dump digits out if not empty DigitCollections are found on StoreGate*
- *It can run directly on digit files*
 - *ReadMuonDigitsFromDigitsFileOptions.py*
- *Or on hit file, performing the digitization on the fly*
 - *ReadMuonDigitsOptions.py*
- *To use it, copy the jobO from MuonDigiExample/share or from ~drebuzzi/public/Tutorial and edit it to set the correct input file name*
- *Run it with*

```
$Dig> athena.py ReadMuonDigitsOptions.py
```

or with

```
$Dig> athena.py ReadMuonDigitsFromDigitFileOptions.py
```



Exercise 14

Digitizing your $Z \rightarrow \mu\mu$ events

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Copy the file *DigitizationConfig.py* under your \sim /Tutorial/Dig directory

```
$Dig> get_files -jo DigitizationConfig.py
```

2. Edit it to set you options, i.e. change
 - the *DetDescrVersion*, set it to 'Rome-Initial'
 - The name of the input file, set it to "Zmumu_Jimmy.pool.root", generated in Exercise 10 (you need the complete path only if the file is not local)
3. run the Atlas Digitization with

```
$Dig> athena DigitizationConfig.py  
Digitization/AtlasDigitization.py
```

4. Use the *ReadMuonDigits* algorithm to check whether *MuonDigits* are properly created
 - Edit *ReadMuonDigitsFromDigitsFile.py* and set the digit filename
 - Run athena with

```
$Dig> athena.py ReadMuonDigitsFromDigitsFile.py
```



Hit Sorting and Pileup

- The hits are created by G4 and collected using the **AthenaHitsVector** container, which is a flat container where they are inserted in a random way (no sorting at the simulation level)
- Before performing digitization, hits from several events which happen in the same or in a nearby bunch crossing are merged
- Hits are read in and pushed into a **TimedHitsVector**
 - Hits (which have their own $G4globalTime = t.o.f$) are provided with an additional time information → time difference between the main crossing and the crossing to which the hits belong
 - Many hit collections can be pushed into the same **TimedHitsVector** (pile-up)
 - Events can be read from more than one file using POOL
- Collections are therefore sorted by detector element (using **HitManagement/TimedHitPtrCollection**)
- The digitization works locally on the sorted hit collections

→ This implementation is able to treat full background (pileup + cavern background)



Muon Digitization

MDT_Digitization

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- The MDT_Digitization algorithm basically converts MDTSimHits into MdtDigits (the conversion Digits \rightarrow RDOs is realized by Cnv which take into account of cabling and readout)

This is done in several steps

1. conversion from drift radius to drift time
2. Calculation of the time structure of the event
 - The drift time (+ ADC charge)
 - The time of flight, by adding to the G4globalTime the drift time
 - The bunch time relative to the current bunch
 - The propagation delay of the signal with respect to the tube readout side
 - The dead time

For a given tube the hits are sorted on time (drift + tof + prop + bunch)

From the first hit the deadtime = time_hit1 + dead-window is initialized (the deadtime is 700ns)

No additional hits will be created if the time of a second hit is smaller than the deadtime

If a hit with $t >$ deadtime is found the deadtime is reset to: deadtime = time_hit2 + dead-window



Muon Digitization

MDT_Digitization (cont'd)

GENERATORS

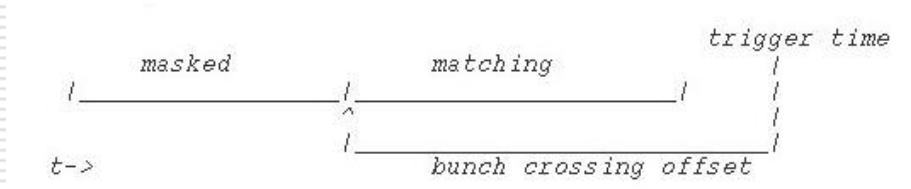
SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

3. Trigger match for the digits



→ Digits coming from hits in the masked window are marked "masked" and will have TDC and ADC counts set to zero

→ Hits in the matching window will produce normal digits

→ If an hit lies outside of both windows no Digit is created

4. Converts total time into TDC counts

■ Optionally the digitizer can check whether the MDTSimHits are ok, corrupted hits (with invalid drift radius, or invalid position on the tube or $\text{tof} > 10 \text{ minTof}$, where the $\text{minTof} = \text{distance from vertex} / \text{light speed}$) are discarded

□ The algorithm has properties which can be changed via jobO

```
File Edit Search Preferences Shell Macro Windows Help
{
  declareProperty("OffsetTDC", m_offsetTDC = 800.);
  declareProperty("ns2TDC", m_ns2TDC = 0.78125);
  declareProperty("ResolutionTDC", m_resTDC = 0.5);
  declareProperty("SignalSpeed", m_signalSpeed = 299.792458);

  declareProperty("InputObjectName", m_inputObjectName = "MDT_Hits");
  declareProperty("OutputObjectName", m_outputObjectName = "mdt_digits" );

  declareProperty("UseAttenuation", m_useAttenuation = false);
  declareProperty("UseTof", m_useTof = true);
  declareProperty("UseProp", m_useProp = true);
  declareProperty("UseTimeWindow", m_useTimeWindow = true);

  declareProperty("BunchCountOffset", m_bunchCountOffset = -200.);
  declareProperty("MatchingWindow", m_matchingWindow = 1000.);
  declareProperty("MaskWindow", m_maskWindow = 700.);
  declareProperty("DeadTime", m_deadTime = 700.);
  declareProperty("CheckSimHits", m_checkMDTSimHits = true );

  declareProperty("DigitizationTool", m_digiToolName = "MDT_Response_DigiTool" );
}
```



MDT_Digitization algorithms

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

For the $r \rightarrow t$ conversion, two tools are available

- **MDT_Response_DigiTool** (default): detailed simulation of MDT response including cluster size fluctuations, diffusion and the ADC response (providing an ADC count)

→ When using this, the rt relation is calculated from hit to hit

- **RT_Relation_DigiTool**: transforms $r \rightarrow t$ + smearing in time using external rt relations, from Garfield, from data (here the ADC count is set to a fixed value)

- If you want to select this → add the following line to the `DigitizationConfig.py` and uncomment it

```
# Uncomment the following line if you want to use MDT
# digitization with external RT relation (default is
# MDT_Response_DigiTool)
# MDT_Digitizer.DigitizationTool = "RT_Relation_DigiTool"
```

→ the default rt relation which is taken here is the `ArCO2.rt` file in `MuonSpectrometer/MuonDigitization/MDT_Digitization/share`

Be extremely careful because the $t \rightarrow r$ conversion in the reconstruction must be selected consequently!



MDT_Digitization

MDT_Digitizer.DigitizationTool='MDT_Response_DigiTool'

GENERATORS

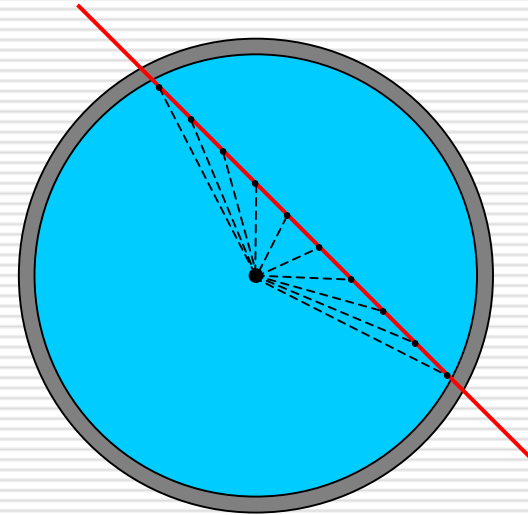
SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- Calculation of particle trajectory
 - Straight line assumed
- Cluster formation
 - Position and size fluctuations
- Propagation of clusters to the wire
 - *rt*-relation
 - Diffusion
 - Lorentz shift
- Signal attenuation in tube
- Simulation of the electronics
 - Amplifier response function
 - Threshold
 - Charge integration



```
File Edit Search Preferences Shell Macro Windows Help
declareProperty("ClusterDensity", m_clusterDensity = 8.5);
declareProperty("Threshold", m_threshold = 20.);
declareProperty("AttenuationLength", m_attenuationLength = 16000);
```

- You can tune some parameters by adding to DigitizationConfig.py

```
ToolSvc = Service( "ToolSvc" )
ToolSvc.MDT_Response_DigiTool.Threshold = 20
```



Exercise 15

Customizing the MDT_Response_DigiTool

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Edit the *DigitizationConfig.py* in order to run the *MDT_Digitization* with the default tool and to set

- *OffsetTDC = 0.*
- *BunchCountOffset = -200*
- *UseTof = FALSE*
- *Signal speed = 18 cm/ns*
- *Electronic threshold = 20 PE*

CTB configuration settings

2. Run the digitization with

```
$Dig> athena DigitizationConfig.py  
Digitization/AtlasDigitization.py
```

→ To set a *MDT_Digitizer* property

```
MDT_Digitizer.UseTof = FALSE
```

→ To set a *Tool* property

```
ToolSvc = Service( "ToolSvc" )  
ToolSvc.MDT_Response_DigiTool.Threshold = 20
```



Customize the Muon Digitization

Customize the *RT_Relation_DigiTool*

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- When using the *RT_Relation_DigiTool*, you can change the *rt* relation, using your customized one
 - check that you have both the *rt* and the *tr* file, in the standard format “à la Calib”
 - check out the package *MDT_Digitization* from your `~/Dig`
 - in its *share* directory, replace the file *ArCO2.rt* with your *rt* file (*r*, *t*, resolution on *r*)
 - compile the *MDT_Digitization* package to allow the *ArCO2.rt* file to be copied to the *InstallArea* (it is set in the `declare_joboptions` in the package `cmt/requirements`)

```
cd MuonSpectrometer/MuonDigitization/MDT_Digitization/  
MDT_Digitization-00-00-*/cmt  
cmt config  
gmake
```



Exercise 16

Digitizing with your rt relation

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. *Edit the DigitizationConfig.py in order to select the RT_Relation_DigiTool*
2. *Check out MDT_Digitization*
3. *In its share directory, replace the file ArCO2.rt with the following rt file*

```
cp ~drebuzzzi/public/Tutorial/ArCO2_ludo.rt  
  MuonSpectrometer/MuonDigitization/MDT_Digitization/MDT_  
  Digitization-*/share
```

4. *Compile MDT_Digitization*
5. *Run the AtlasDigitization prepending DigitizationConfig.py*

→ *to pass to the reconstruction the same tr relation, copy this line in your reconstruction jobO*

```
MuonTBCalibrationSvc.RT_InputFiles = [ "/afs/cern.ch/user/d/  
drebuzzzi/public/Tutorial/ArCO2_ludo.tr" ]
```



MuonDigitization

XxxDigitValidation algorithms

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- *XxxDigitValidation algorithms* → known track position and truth objects are compared to digit position and parameters and the residual distributions are studied
 - During the digitization, a separate object is recorded and can be persisted together with the RowData object, the MuonSimData. It maintains the link to the hit at the Digit/RDO level
 - **MDTs**: drift radius residual ($r_{\text{calc}} - r_{\text{true}}$) where r_{true} is coming from the hits information
 - **CSCs**: track distance from the strip with the highest charge – clusterization validation
 - **RPCs**: track distance to the nearest RpcDigit - fixed error to the simID which propagated to RPC_Digitizer – clusterization validation
 - **TGCs**: hit positions in TgcSimHit compared to those
-
- *The output of each validation algorithm is an ntuple which contains the main digit parameters*



Documentation and useful links

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- **Atlas Digitization web page**

<http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/simulation/digitization>

- **A Muon Digitization note in preparation**

“Geant4 Muon Digitization in the ATHENA Framework”,
D. Rebutzi, K.A. Assamagan, A. Di Simone, Y. Hasegawa, N. Van Eldik



PART IV: Running in batch

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

What you should expect to learn...

- How to use the CERN LSF*
- How to use ASK to submit a job*
- Submitting jobs with ASK for event generation, simulation and digitization on LSF batch queues*
- Running in batch using the “Rome workshop” scripts*
 - *Simulation/SimulationOptionsRome/scripts*
 - *Simulation/Digitization/scripts*



Use of CERN LSF

- *The CERN batch farm (lxbatch) is simple to use because it has access to same file systems as lxplus, the interactive nodes*
 - *you can copy files from your home directory to the batch machine for processing and return the results with simple shell commands*
- *You can not login on lxbatch machines. You submit jobs to them into a particular queue, based on the required amount of CPU time (one week,two days, one day, etc.), using LSF*
- *The batch job typically consists of a shell script which 1) sets up the environment, 2) copies auxiliary files, and 3) runs the executable*

□ *How to monitor the batch jobs*

```
$> bqueues
```

→ *To list all the available batch queues*

```
$> bjobs
```

→ *To retrieve the status of the jobs*

```
$> bpeek #JOBID
```

→ *To look at the JOBID output. JOBID identifier is obtained from bjobs*

```
$> bpeek -f #JOBID
```

→ *“-f” allows to follow the tail*



Submitting jobs with ASK

- ASK can be used to submit jobs on LSF
- Create a work dir under ~/Tutorial

```
$> mkdir Batch
```

and create the following five directories under it

```
$> mkdir batch  
$> mkdir scripts  
$> mkdir log  
$> mkdir output  
$> mkdir pool
```

- Copy the ASK based submission script `tutorialjob.sh` in your ~/Batch dir from `~drebuzzzi/public/Tutorial` and make it executable
- The script takes one argument, the ASK script which is to be used and 1) copies the required files in the work dir on the batch machine, 2) starts ASK, 3) executes the given script, which can contain any command as you issue on the `ask>` prompt
- Submit the job with ASK script `myscript.py` into the batch queue with the command (for example “1nh” one normalized hour)

```
$> bsub -o log/mylog.log -q 1nh tutorialjob.sh myscript.py
```



Running in batch using ASK

Generators

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- *To run a generation job in batch on CERN LSF, prepare an ASK script with the following contents or copy it from my ~/public/Tutorial*

```
# setup the run-time environment
runtime( '10.0.2' )
# get the options file for this run
getfiles( '$JOBDIR/scripts/jobOptions.pythia.py' )
# process the actual generators job
run( '-b run/jobOptions.pythia.py' )
# save the results
copyfile( 'run/pythia.pool.root', '$JOBDIR/output/.' )
# update the POOL file → Set here the name of your output file
import AthPOOL
AthPOOL.merge( '$MYPOOLFILE' )
```

*Name the scrips Generators.py and copy it in the ~/scripts dir.
Copy also your local, modified jobO file into the same directory*

- *Run your batch job from your work dir with*

```
$> bsub -o log/mylog.log -q lnh tutorialjob.sh -d 'pwd'  
Generators.py
```



↔ Exercise 17

Submitting a generator job with ASK

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Submit a generation job to Ixplus using ASK interface
2. As jobO for the simulation, use the Pythia jobO `jobOptions.pythia.py` for $Z \rightarrow \mu\mu$ created in the first part of this tutorial
 - Checked that it writes events in POOL and that the output name is correctly set in `Generator.py`

→ After the job is finished, you will find a log file in the `~/log` directory and a POOL output file in the `~/pool` directory. This output can be used as input for the simulation



Running in batch using ASK

Simulation

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- ❑ To run a simulation job in batch on CERN LSF, prepare an ASK script, *Simulation.py* (or copy it from `~drebuzzi/public/Tutorial`), with the following contents and copy it in the `~/scripts` directory

```
# setup the run-time environment
runtime( '10.0.2' )
# get the options file for this run
getfiles('$JOBDIR/scripts/jobOptions.G4Atlas_Sim.py' )
# use restricted cuts in eta
getfiles('$JOBDIR/scripts/Filters.mac' )
# get the data file
getfiles('$JOBDIR/output/pythia.pool.root' )
# actual simulation job
run( '-b run/jobOptions.G4Atlas_Sim.py' )
# save the results
copyfile( 'run/g4hits.pool.root', '$JOBDIR/output/.' )
# update the POOL file
import AthPOOL AthPOOL.merge( '$MYPOOLFILE' )
```

→ Set here the name of your output file

- ❑ Copy your simulation jobO and the *Filters.mac* file into the same dir. Copy the *PoolFileCatalog.xml* in the `~/pool` dir and the generated event file in the `~/output` dir
- ❑ Run your batch job from your work dir with

```
$> bsub -o log/simulation.log -q 1nd tutorialjob.sh -d `pwd`
Simulation.py
```



↔ Exercise 18

Submitting a simulation job with ASK

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- Submit a simulation job to Ixplus using ASK interface
- As input for the simulation, use the $Z \rightarrow \mu\mu$ events created in the previous exercise
- As simulation jobO, use `jobOptions.AtlasGeoG4Sim.py`
 - Retrieve it with the `get_files` command, together with the `Filters.mac` file

Special filter mechanisms can be applied on the HepMC event before undergoing the simulation

→ With this filter, the simulation traces all particles between pseudorapidity of -7 and +7. To speed up the job, it is possible to change the range in eta.

```
/Filters/Vertex/toggle ON  
/Filters/Vertex/Spread 0.01 0.01 50 mm  
/Filters/EtaPhiFilter/toggle ON  
/Filters/EtaPhiFilter/EtaInterval -7. 7.
```

→ After the job is finished, you will find a log file in the `~/log` directory and a POOL output file in the `~/pool` directory. This output can readily be used as input for the digitization



Running in batch using ASK

Digitization

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- To run a digitization job in batch on CERN LSF, prepare an ASK script, *Digitization.py* (or copy it from `~drebuzzi/public/Tutorial`), with the following contents and copy it in the `~/scripts` dir

```
# setup the run-time environment
runtime( '9.0.3' )
# retrieve the data file
getfiles( '$JOBDIR/output/g4hits.pool.root' )
# actual simulation job
run( '-b -c \'PoolHitsInput = [ "g4hits.pool.root" ];
PoolRDOOutput = "g4digi.pool.root";
EvtMax = -1\' Digitization/AtlasDigitization.py' )
# save the results
copyfile( 'run/g4digi.pool.root', '$JOBDIR/output/.' )
# update the POOL file → Set here the name of your output file
import AthPOOL AthPOOL.merge( '$MYPOLFILE' )
```

- Copy your digitization jobO into the same dir. Copy the *PoolFileCatalog.xml* in the `~/pool` dir and the simulated event file in the `~/output` dir
- Run your batch job from your work dir with

```
$> bsub -o log/digitization.log -q lnh tutorialjob.sh -d
'pwd' Digitization.py
```



↔ Exercise 19

Submitting a digitization job with ASK

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- *Submit a digitization job to lxplus using ASK interface*
- *As input, use the simulated $Z \rightarrow \mu\mu$ events created in the previous exercise*
- *As jobO for the digitization, use AtlasDigitization.py*

→ *After the job is finished, you will find a log file in the ~/log directory and a POOL output file in the ~/pool directory. This output can readily be used as input for the reconstruction*



Simulation scripts

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- The *Simulation/SimulationOptionsRome* package is a collection of scripts (transformations and wrappers) to run the G4 simulation (with releases $\geq 9.0.2$) and the “Rome workshop” configuration, for submission to LSF batch

Current directory: [\[atlas\]](#) / [offline](#) / [Simulation](#) / [SimulationOptionsRome](#) / [script](#)
Files shown: 11

File	Rev.	Age	Author	Last log entry
Attic/ [show contents]				
LSFwrapper_RomeSimulation_EtaRange	1.8	2 months	nairz	modifications for 9.0.4
LSFwrapper_RomeSimulation_SinglePart	1.2	4 weeks	nectar	fixes in the single-particles scripts
LSFwrapper_RomeSimulation_SinglePart_dEdx	1.1	7 weeks	nairz	added scripts for single-particle simulation
LSFwrapper_RomeSimulation_SinglePart_dEdx_noVx	1.1	6 weeks	nectar	added single-particle scripts with no vertex spread
LSFwrapper_RomeSimulation_standard	1.11	2 months	nairz	modifications for 9.0.4
rome_g4sim.CaloCalibration.trf	1.4	2 weeks	nairz	modified handling of magn. fieldmap in trfs.
rome_g4sim.etarange.trf	1.8	2 weeks	nairz	modified handling of magn. fieldmap in trfs.
rome_g4sim.partgen.dedx.novx.trf	1.4	2 days	nairz	fixed typo in single-particle simulation scripts
rome_g4sim.partgen.dedx.trf	1.5	2 days	nairz	fixed typo in single-particle simulation scripts
rome_g4sim.partgen.trf	1.4	2 days	nairz	fixed typo in single-particle simulation scripts
rome_g4sim.standard.trf	1.8	2 weeks	nairz	modified handling of magn. fieldmap in trfs.

- Many transformation (and wrappers) available, according to different configurations for the simulation

- Reading a generated file or using ParticleGenerator
- Applying an eta filter
- No vertex spread
- dE/dx only
- Calorimeter calibration



Exercise 20

submit a simulation script job

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- Check out the *Simulation/SimulationOptionsRome* package
- Copy the *rome.g4sim.partgen.trf* transformation and the *LSFwrapper_RomeSimulation_SinglePart* wrapper from *Simulation/SimulationOptionsRome/scripts* to your *~/Batch* dir
- Customize the wrapper in order to
 - simulate 100 single μ^- at 6 GeV, in the $|\eta| < 1.2$ range
- Save the output on your castor dir

→ Maybe you do not know, but having an account on *lxplus*, gives you a castor space under

/castor/cern.ch/user/x/username/

Where *x* stands for your username initial letter

→ When using */castor*, you should use the castor syntax, i.e. */rfio;*
rfdir, rfrename, rfcp, etc..

- Submit the simulation job on *lxbatch* with

```
bsub -q lnd LSFwrapper_RomeSimulation_SinglePart
```



Digitization scripts

GENERATORS

SIMULATION

DIGITIZATION

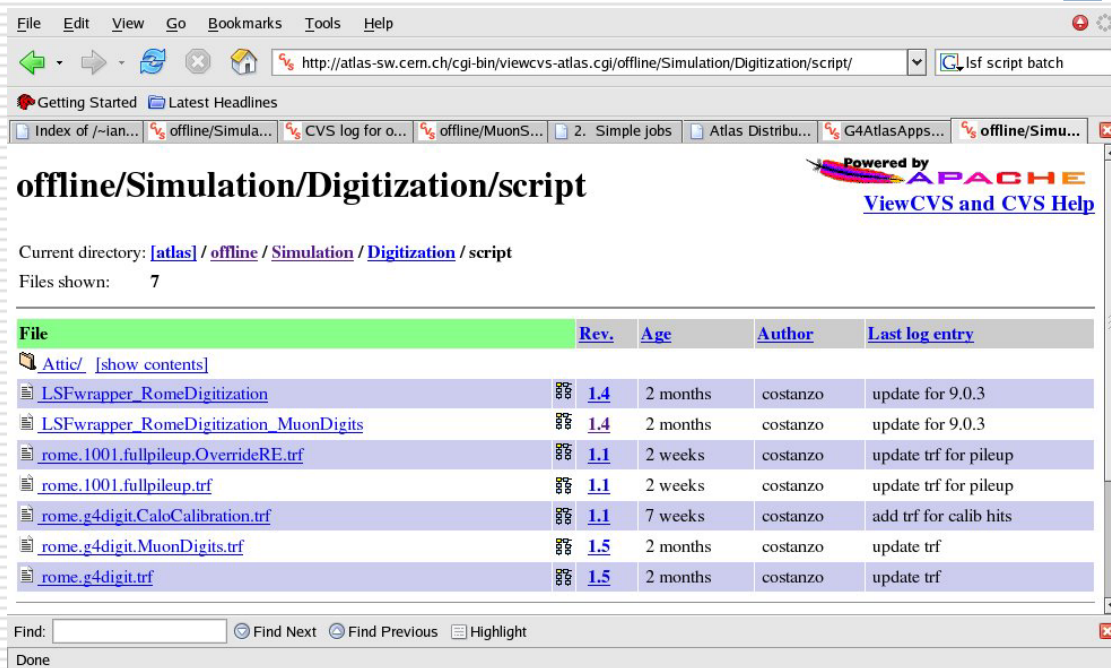
BATCH RUNNING

APPENDICES

- The *Simulation/Digitization/script* directory contains a collection of scripts (transformations and wrappers) to run the digitization (with releases > 9.0.2) and the “Rome workshop” configuration, for submission to LSF batch

- Different transformations and wrappers

- to have *MuonDigits* or *RDOs* as output from the *MuonDigitization*
- to run the pileup (with release 10.0.1)
- To run with *calo* calibrations



Current directory: [\[atlas\]](#) / [offline](#) / [Simulation](#) / [Digitization](#) / [script](#)

Files shown: 7

File	Rev.	Age	Author	Last log entry
Attic/ [show contents]				
LSFWrapper_RomeDigitization	1.4	2 months	costanzo	update for 9.0.3
LSFWrapper_RomeDigitization_MuonDigits	1.4	2 months	costanzo	update for 9.0.3
rome.1001.fullpileup.OverrideRE.trf	1.1	2 weeks	costanzo	update trf for pileup
rome.1001.fullpileup.trf	1.1	2 weeks	costanzo	update trf for pileup
rome.g4digit.CaloCalibration.trf	1.1	7 weeks	costanzo	add trf for calib hits
rome.g4digit.MuonDigits.trf	1.5	2 months	costanzo	update trf
rome.g4digit.trf	1.5	2 months	costanzo	update trf



Exercise 21

submit a digitization script job

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- ❑ Check out the Simulation/Digitization package
- ❑ Copy the `rome.g4digit.MuonDigits.trf` transformation and the `LSFwrapper_RomeDigitization_MuonDigits` wrapper from `Simulation/Digitization/scripts` to your `~/Batch` dir
- ❑ Make the transformation script executable
- ❑ Modify the wrapper in order read the single μ file simulated in the previous exercise and to save the output on your castor dir
- ❑ Submit the simulation job on `lxbatch` with

```
bsub -q 2nh LSFwrapper_RomeDigitization_MuonDigits
```

→ Check in the `PoolFileCatalog` that you have the entry for your input file or forget about it if you are running with a version $\geq 10.0.1$



Digitization + pileup scripts

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

- *It is possible to run ATLAS pile-up jobs with full background, i.e. minimum-bias and cavern, on the CERN LSF batch farm, using Release 10.0.1 and the 'Rome-Initial' layout.*
- *minimum-bias and cavern background can be superimposed to signal events*
 - *Up to now tested only with single muons and $Z \rightarrow \mu\mu$ events*
 - *Signal and background events (SimHits) are read from simulated files*
- *There are different luminosity scenarios available (i.e. lumi01 = 10^{33} cm⁻²s⁻²)*
- *The pileup script 1) mixes signal and background hits, 2) performs digitization, and 3) writes out the resulting RDOs to a POOL output file*



↔ Exercise 22

submit a pileup with full background script job

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

1. Copy the wrapper script `LSFwrapper_RomePileUp` from `Simulation/Digitization/script` and the core pile-up script ('transformation') `rome.fullpileup.trf` to your `~/Batch` dir
2. make the scripts executable
3. modify the wrapper script in order to set the `OUTPUTDIR` to the desired output directory
4. submit the job to LSF, with

```
bsub -q 1nd -R "select[maxmem>1000]" -o logfile.txt  
LSFwrapper_RomePileUp
```

→ the job will require memory of about 800 MB, therefore the "-R" option in the submission command

6. Modify the wrapper in order read the single μ file simulated in the previous exercise
7. Submit the simulation job on `lxbatch` with

```
bsub -q 2nh LSFwrapper_RomeDigitization_MuonDigits
```

8. Save the output on your `castor` dir



Background files

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

□ Background files (both minimum-bias and cavern), simulated with Rel. 9.0.4, $|\eta| < 6$, are available at the following locations:

□ **minimum-bias files**

- `/castor/cern.ch/user/c/costanzo/minbias`
(100 files a 50 events; from Rome production)
- `/castor/cern.ch/atlas/project/rome_phys_ws/preprod/romesim904/minbias`
(20 files a 200 events; from pre-production tests)

□ **cavern-background files** ('sf' = 'safety factor'; 'sf01' = nominal background, 'sf02' = twice nominal background, etc.)

- `/castor/cern.ch/atlas/project/muon/cavern_background/simul/rome/sf01/data`
(140 files a 200 events; partition number > 1000)
- `/castor/cern.ch/atlas/project/muon/cavern_background/simul/rome/sf02/data`
(70 files a 200 events; partition number > 1000)
- `/castor/cern.ch/atlas/project/muon/cavern_background/simul/rome/sf05/data`
(28 files a 200 events; partition number > 1000)
- `/castor/cern.ch/atlas/project/muon/cavern_background/simul/rome/sf10/data`
(14 files a 200 events; partition number > 1000)

□ **POOL file ID's** can be found in

`/afs/cern.ch/atlas/project/dc1/muon/cavern_background/PoolFileCatalog_all.xml`

□ **Pre-assembled background file lists** (needed as input to the transformation and to be defined in the wrapper script) can be found in

`/afs/cern.ch/atlas/project/dc1/muon/cavern_background/bkg_files`



Useful links

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

□ **LSF web page**

<http://batch.web.cern.ch/batch/>

□ **Atlas digitization + pileup web page**

http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/simulation/geant4/how2_run_pileup_1001.html



PART V: Appendices

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

What you should expect to find here...

- Muon hits and digits data members*
- Acknowledgements*



MuonSimHits

GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES

CSCSimHits

```
int m_CSCid;  
double m_globalTime;  
double m_energyDeposit;  
Hep3Vector m_hitStart;  
Hep3Vector m_hitEnd;  
int m_particleID;  
double m_meanTime;
```

RPCSimHits

```
int m_RPCid;  
double m_globalTime;  
Hep3Vector m_localPosition;  
double m_meanTime;
```

MDTSimHits

```
HitID m_MDTid;  
double m_globalTime;  
double m_driftRadius;  
Hep3Vector m_localPosition;  
double m_meanTime;
```

TGCSimHits

```
int m_TGCid;  
double m_globalTime;  
Hep3Vector m_localPosition;  
Hep3Vector  
m_localDireCos;  
double m_meanTime;
```



MuonDigits

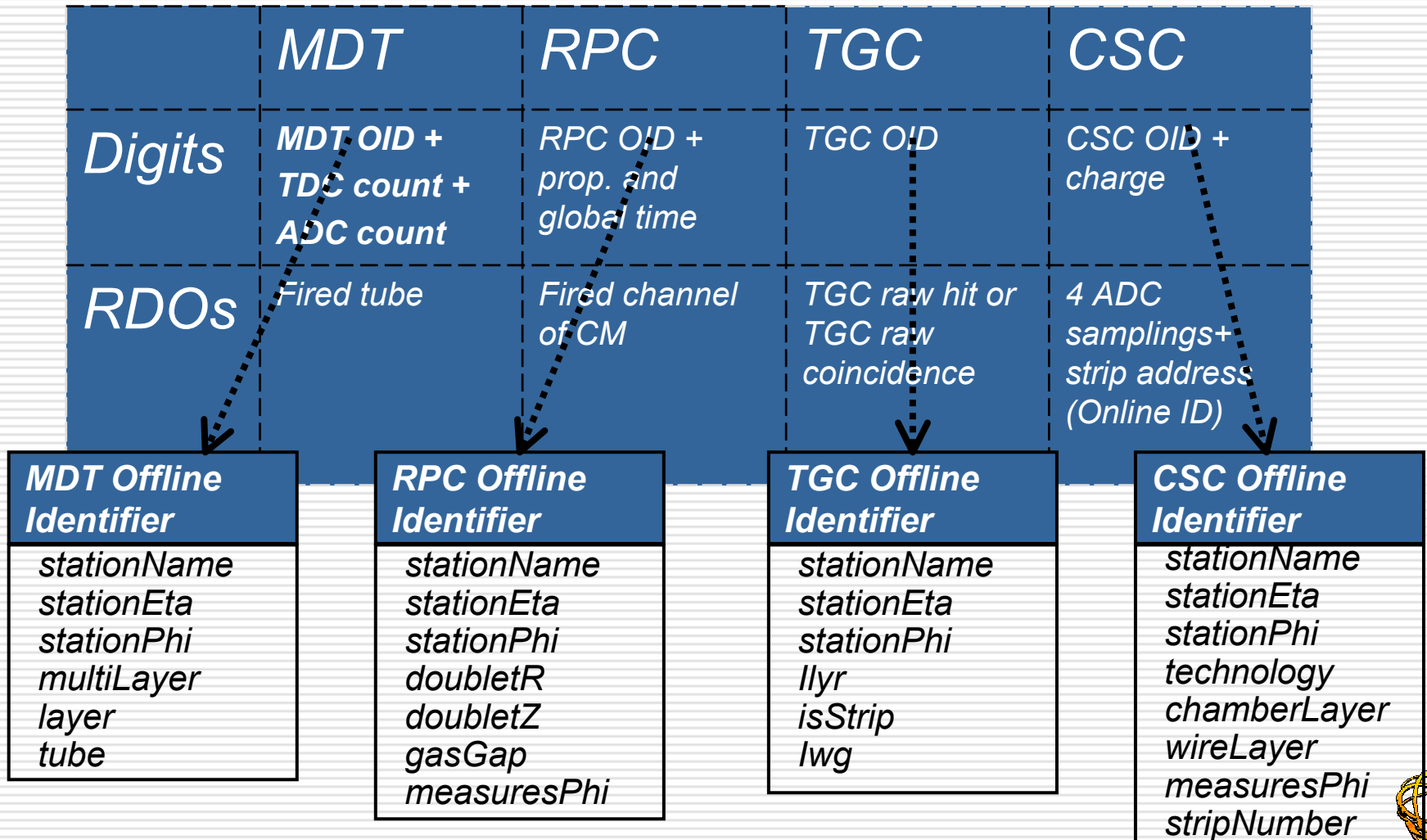
GENERATORS

SIMULATION

DIGITIZATION

BATCH RUNNING

APPENDICES



Acknowledgements

Special thanks!

Andrea Dell'Acqua

Manuel Gallas

Wim Lavrijssen

Stefano Rosati

Ketevi A. Assamagan

Davide Costanzo

