Configuring Geant4 applications with Python (ATLAS approach)

Manuel Gallas, Andrea Dell'Acqua

LCG Applications Area meeting

13 July 2005, CERN

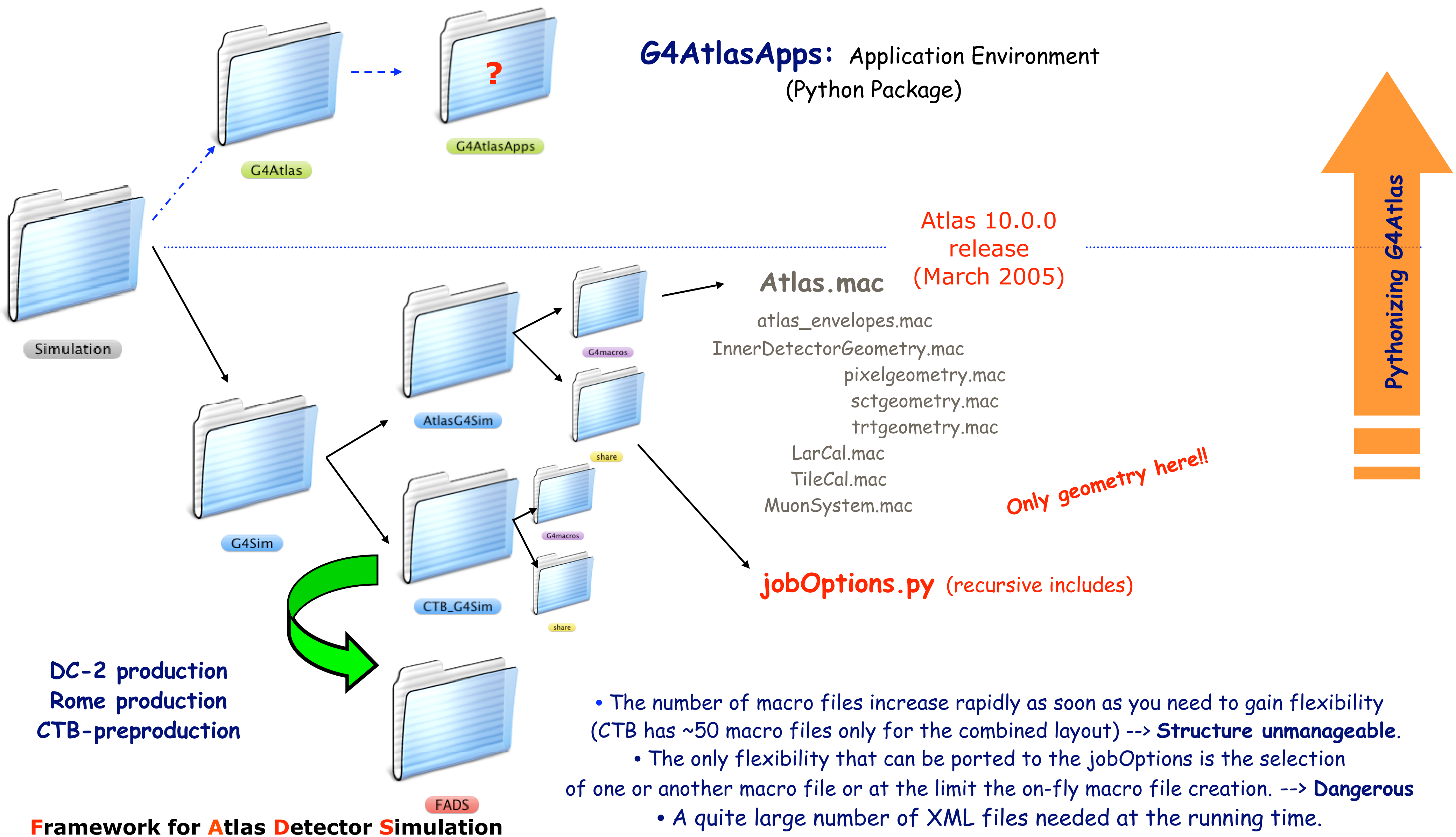
Outline:

- Some questions to begin with
- Global picture
- What do we look for?
- The result product
- Some use-cases (examples)
- Conclusions

Some questions to begin with:

- **What?** be able to configure different Geant4 Applications from the Python prompt and provide an interactive approach. "Configure" in a sense in which the end-user can go from very top configuration (simulation options and flags) to the detail and be able to build and customize the simulation [we expect different use-cases].
- **Why?**
  - The ATLAS Athena framework provides a Python prompt and the configuration of the jobs (generation, simulation, digitization, reconstruction, physics analysis ...) is done through Python scripts ("jobOptions"). So it makes sense to provide a way to not only launch but also configure the simulation jobs.
  - The standard "G4 macros" are not well integrated in this Python infrastructure:
    - they are txt files that we do not want to parse, copy here or there or build on the fly.
    - The number of these macro files has an uncontrolled growing tendency.
    - At the end the interactivity becomes a heavy editing activity on these macro files.
- **How?** This is the topic of the talk and covers the work done by ***Andrea Dell'Acqua*** and myself during the last months to solve the equation : **Geant4+PyLCGDict+Python+ATLAS specific = PyG4AtlasApps** a particular (and not for this more easy!) case of the general case: **Geant4+PyLCGDict+Python=PyG4Apps**
- **Where?** This approach is working for the different ATLAS Geant4 simulation applications (2004 Combined Test Beam in GRID production since May 2005 and ~ 5 Million events, ATLAS full simulations for the commissioning and cosmic studies by ***Andrea Di Simone***)

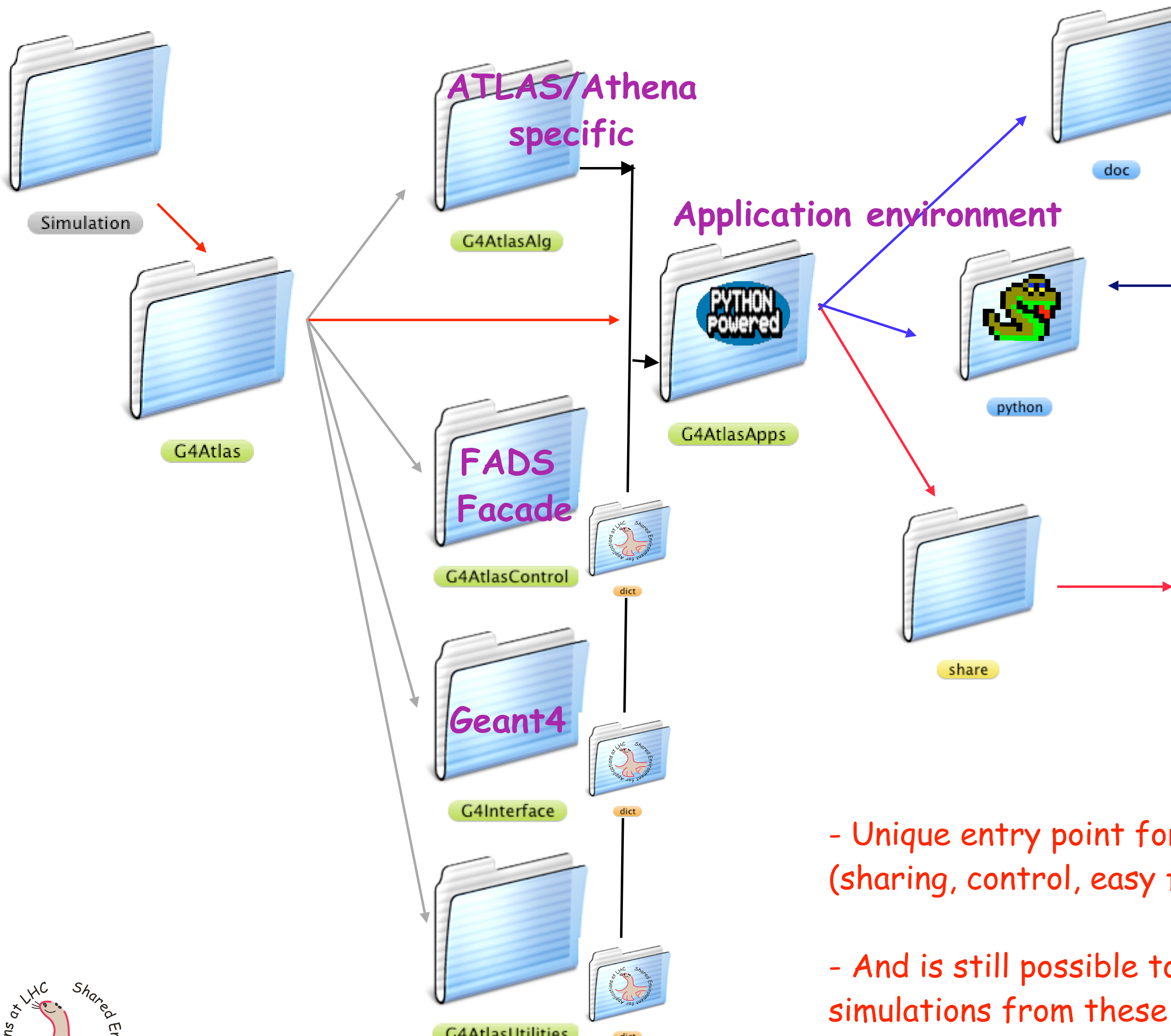
Global picture: ATLAS simulation road map (I = migration)



- The number of macro files increase rapidly as soon as you need to gain flexibility (CTB has ~50 macro files only for the combined layout) --> **Structure unmanageable**.
- The only flexibility that can be ported to the jobOptions is the selection of one or another macro file or at the limit the on-fly macro file creation. --> **Dangerous**
- A quite large number of XML files needed at the running time.
- The interactive configuration of the simulation job and navigation in the Athena prompt is complicated (ex: Which are the cuts am I applying?).

Manuel Gallas
CERN PH-SFT

Global picture: ATLAS simulation road map (II = new life)



Documentation almost for free

Python modules in the $PHYTONPATH
can be accessed from everywhere

jobOptionG4Atlas_Sim.py
jobOptionG4Cosmic.py
jobOptionG4InDetcosmic.py
jobOptionsG4CTB_Sim.py
....
test_ATLAS-evgenmu_pt5_eta60.py
test_ATLAS-singleE_ParLArBarrel.py
....

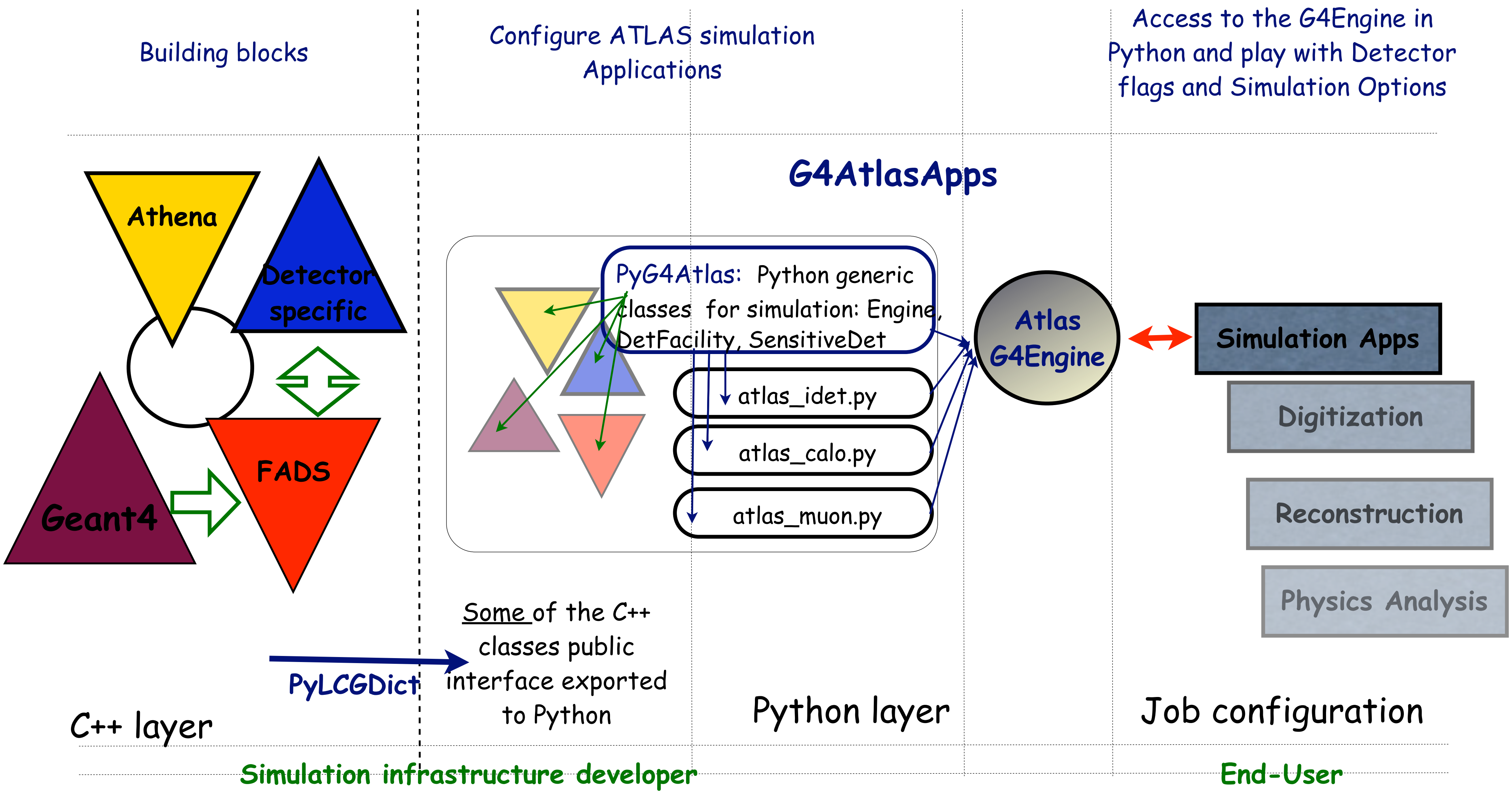
- Unique entry point for all the simulations apps (sharing, control, easy for the end user)

- And is still possible to inspect and customize the simulations from these top jobOptions

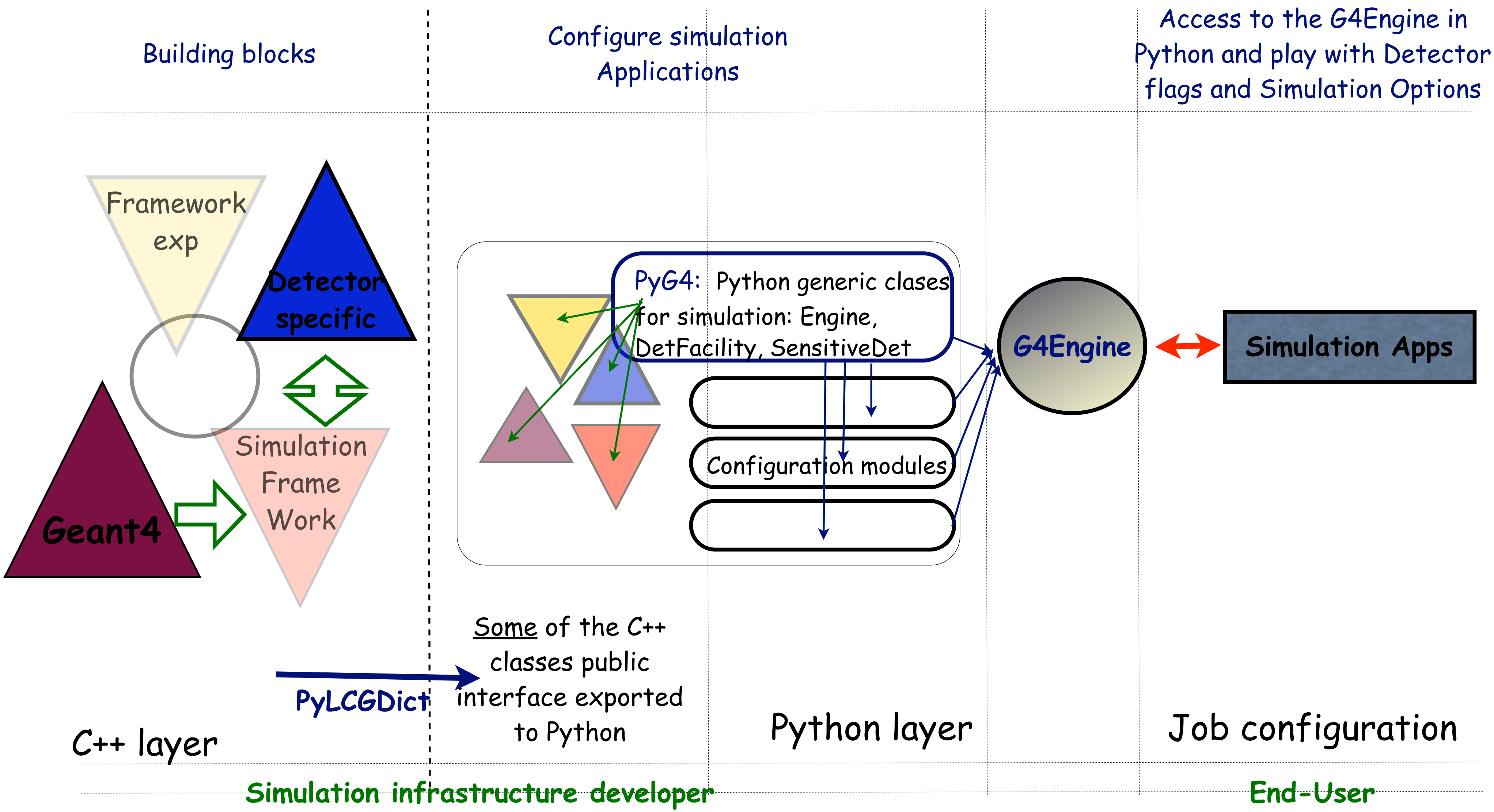
SEAL: the PyLCGDict mechanism allows for straightforward exporting of C++ classes into Python

Manuel Gallas
CERN PH-SFT

Global picture: (III = general view)



Global picture: (IV = generic view)



What do we look for?

1. **Provide user-functionality for the simulation jobs and of course <u>make happy all users</u> (difficult!!),** but let us see what they may need:

Entry Level user

- **User at Entry-Level**
  (entry-point JobOption.py and use of flags)
  - Switch on/off detectors (<u>DetFlags</u>) and build the simulation automatically in accordance.
  - Run the simulation with run conditions.
  - Run different geometries for the sub-detectors.
  - Access to different physics lists/regions/cuts.
  - Of course select particle/energy/magnetic field

Advanced user

- **User that wants to configure/customize the application**
  (entry-point JobOption.py)
  - Possibility to modify the existing simulation from the jobOption.py (no needed to touch G4AtlasApps package, only access to the modules defined there).
  - To do what?:
    - change the position of the detectors, (particular studies, misalignments…)
    - add materials, scintillators, define new regions..

Simu Infrastructure developer

- **Advanced user which implements the simulation entities**
  (simulation infrastructure)
  - Navigate the complete set of simulation objects (detector facilities, sensitive detectors, cuts, positions, envelopes hierarchy)
  - Use and reuse as much as possible python objects already defined and speed up the setup of a new simulation

Production user

- **Production user:** (production team)
  - Flexibility to run what the Physics Community requires but avoid complicated scripts to customize the simulation on-fly.
  - Hits/Digits jobs (same running conditions)

2. **Minimize the spread of configuration files/ modules.**
   - we really want to forget the "mac" files and their uncontrolled growing.
3. **Full Atlas/CTB/…. and other simulation must be run in the same way and in the same place, sharing as much as possible.**
   - users may need to run both simulations (or other configurations) and the change must be in selecting the right jobOption.py not the package.
4. **Flags mechanism under control:**
   - users do not need to be exposed to more flags than needed and should know where to find them (documentation).
   - but the flags mechanism has to be flexible enough to define new configurations (actual full Atlas and CTB simulation have common and different flags.
5. **A some point we maybe need to be able to run getting the run conditions from a DB.**

The result product

Geant4+PyLCGDict+Python+ATLAS specific = PyG4Atlas

PyG4Atlas is a Python interface to interact from the Athena Python prompt with FADS (Framework for the ATLAS Detector Simulation) and G4 .

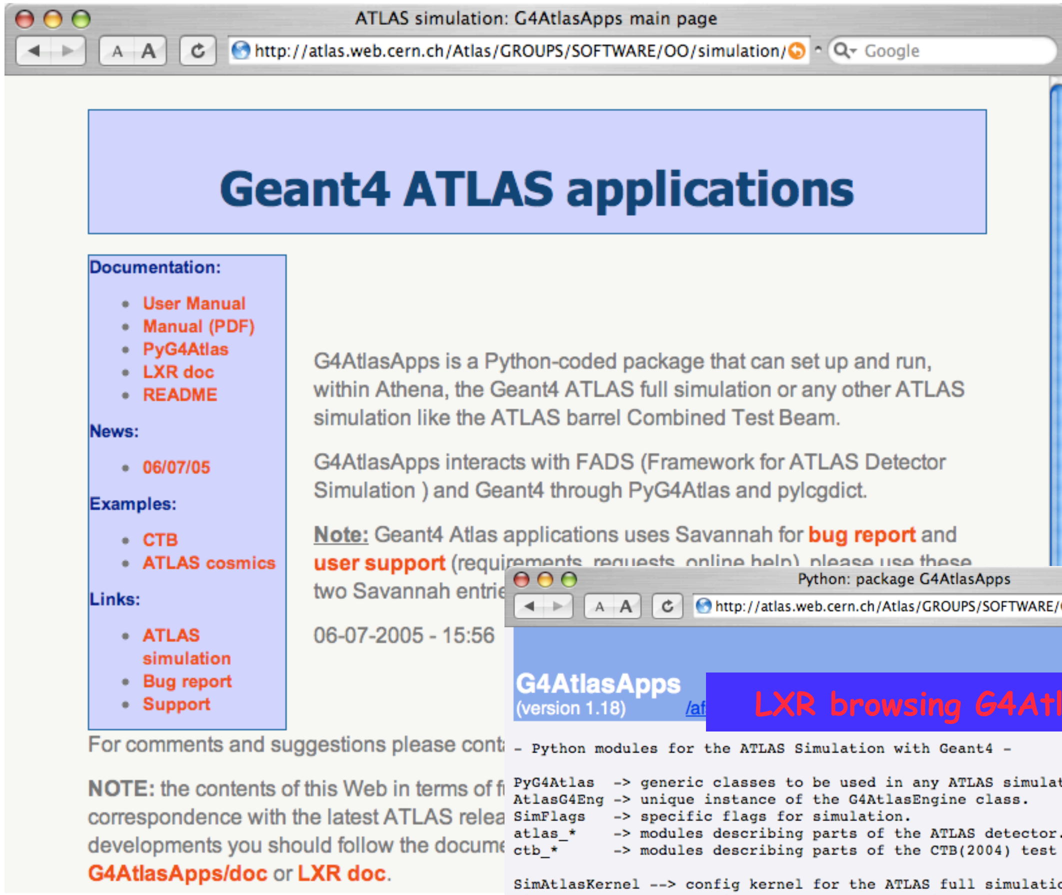
> **PyG4Atlas** defines the python classes for: DetectorFacilities, Sensitive Detectors, Mctruth strategies, PhysicsRegions and Cuts, PhysicsLists,UserActions…
>
> **PyG4Atlas.G4AtlasEngine** puts everything together (it should be possible to access any python object involved) and takes care of the different phases of initialization, log-service, etc…
>
> **PyG4Atlas** always does a selective import of python modules, lib, dictionaries based on the user requirements.

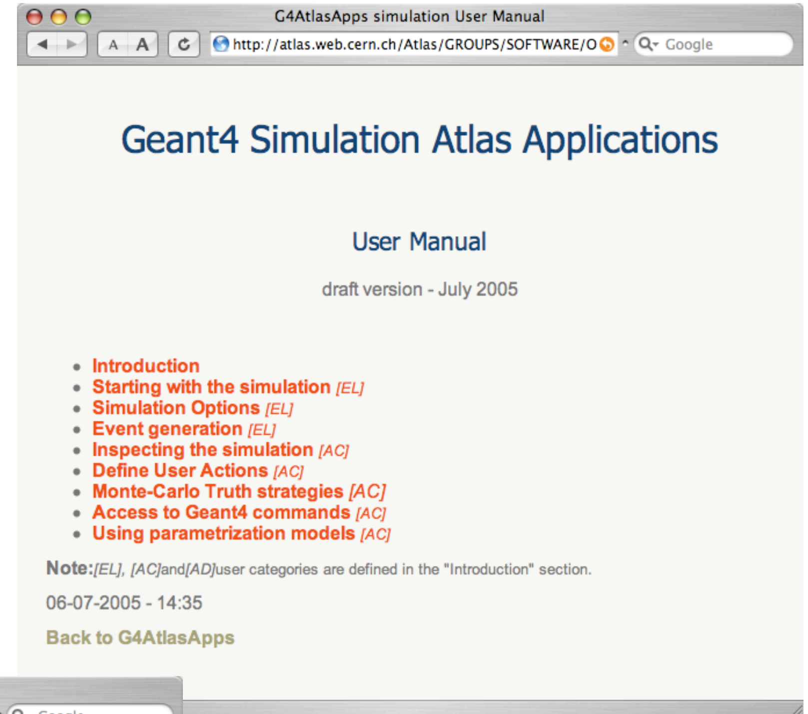
G4AtlasApps is an application environment package that provides PyG4Atlas interface and a set of pre-configured ATLAS simulation cases:

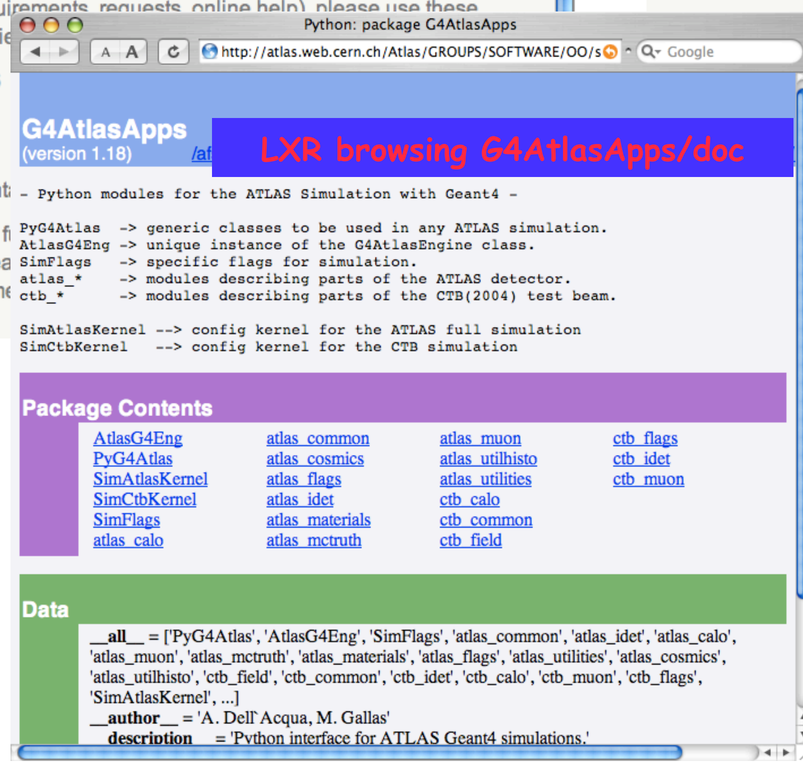
- **full ATLAS, commissioning**
- **full ATLAS cosmic studies**
- **Inner Detector cosmic studies,**
- **CTB test beam (different layouts)**

that the user can customize using SimFlags (at first order) or by accessing the Python simulation objects.

The result product







Some use cases(1): Entry level user

Configuration of the simulation job

Terminal — ssh — 66x49

```
#==================================================================
#
# Job options file for Geant4 Simulations
#
# CTB_G4Sim: CTB (2004) simulation
#
#==================================================================

#--- Detector flags -------------------------------------------------
from AthenaCommon.DetFlags import DetFlags
# - Select detectors
DetFlags.ID_setOn()
#DetFlags.pixel_setOff()
DetFlags.Calo_setOn()
DetFlags.LAr_setOff()
DetFlags.em_setOff()
DetFlags.Tile_setOn()
DetFlags.Muon_setOn()
# - MCTruth
DetFlags.simulate.Truth_setOn()

#--- Simulation flags -----------------------------------------------
from G4AtlasApps.SimFlags import SimFlags
SimFlags.import_Flags('ctb_flags')  # - specfic CTB flags

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

# - Option1: run using specfic CTB flags for combined layout
SimFlags.SimLayout.set_Value('ctbh8_combined')
SimFlags.LArEMBenergyCor.set_On()
SimFlags.Eta.set_Value(0.2)
SimFlags.MagnetMBPSIDBz.set_Value(-1.4)

# - Option2: run for LAr material studies
#SimFlags.SimLayout.set_Value('ctbh8_lar-material')
#SimFlags.LArEMBenergyCor.set_On()
#SimFlags.Eta.set_Value(0.4)
#SimFlags.LArMaterialAluThickness.set_Value(25.)

# - Option3: run using run-conditions for the CTB runs,
#           demo DB (example with run 242)
#SimFlags.RunConditions.set_Value('CtbRunConditions')
#SimFlags.RunNumber.set_Value(242)

#--- Generator flags ------------------------------------------------
SimFlags.Seeds.set_Value('SINGLE 2000160768 643921183')
SimFlags.ParticlePDG.set_Value('11')
SimFlags.Energy.set_Value(50000)
                                                1,1           Top
```

ATLAS CTB test beam

Just an example for a ATLAS CTB simulation user:

- Uses detector flags (detectors on/off).
- Uses Simulation flags (PersistencyHit,
  Kinematics mode,
- Uses specific CTB simulation flags:
  SimFlags.SimLayout.setValue(cth8_combined
  cth8_photon
  ctb8_calibration
  ctb8_lar-material)
  SimFlags.Eta.setValue(0.2)
  SimFlags.IdetOracleTag.setValue('InnerDetector-CTB-05')

- Uses run conditions and wants run number 242
(the same conditions must be passed to digitization job)

Some use cases(1): Entry level user

Configuration of the simulation job

Terminal — ssh — 64x44

```
#================================================================
#
# Job options file for Geant4 Simulations
#
# Atlas simulation
#
#================================================================

#--- Detector flags -------------------------------------------
from AthenaCommon.DetFlags import DetFlags
# - Select detectors
DetFlags.ID_setOn()
DetFlags.Calo_setOn()
DetFlags.Muon_setOn()
#
DetFlags.simulate.Truth_setOn()
#--- Simulation flags -----------------------------------------
from G4AtlasApps.SimFlags import SimFlags
SimFlags.import_Flags('atlas_flags')

SimFlags.SimLayout.set_Value('Rome-Final')
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')
#SimFlags.EvgenInput.set_Value("rfio:/castor/cern.ch/atlas/proje
ct/dc2/preprod/evgen805/dc2.002930.mu_pt30_eta320.evgen805/data/
dc2.002930.mu_pt30_eta320.evgen805._0001.pool.root")

# - uses a given generator
#SimFlags.KinematicsMode.set_Value('EventGenerator')
#SimFlags.GeneratorPath.set_Value('G4AtlasApps/Z-mumuOptions.py'
)
#SimFlags.GeneratorPath.set_Value('GeneratorOptionsRome/rome.004
201.JimmyZee.py')

# No magnetic field
#SimFlags.MagneticField.set_Value(False)
                                              1,1           Top
```

ATLAS full simulation

Terminal — ssh — 67x28

```
#==================================================================
#
# Job options file for Geant4 Simulations
#
# Atlas+Cavern cosmic simulation
#
#==================================================================

#--- Detector flags ----------------------------------------------
from AthenaCommon.DetFlags import DetFlags
# - Select detectors
DetFlags.ID_setOn()
DetFlags.Calo_setOn()
DetFlags.Muon_setOn()
#
DetFlags.simulate.Truth_setOn()
#--- Simulation flags --------------------------------------------
from G4AtlasApps.SimFlags import SimFlags
SimFlags.import_Flags('atlas_flags')

SimFlags.SimLayout.set_Value('Commissioning-1')
SimFlags.PersistencyHit.set_Value('atlas_MyOutputFile.root')

# - uses a given generator
SimFlags.KinematicsMode.set_Value('EventGenerator')
SimFlags.GeneratorPath.set_Value('G4AtlasApps/CosmicGenerator.py')
                                                 1,1           Top
```

ATLAS full simulation cosmic rays

For the user the three cases we have described here behaves in the same way. He can run all of them in the same test directory and the way in which the jobs are configured is the same.

Some use cases(1): Entry Level user

Terminal — ssh — 52x54

```
athena> SimFlags.SimLayout.print_Flag()
G4AtlasApps.SimFlags:: SimLayout True ctbh8_combined
athena> Wv=AtlasG4Eng.G4Eng.Dict.get('volume_world')
athena> Wv.listrecursive_DetFacility()
. MUON
.. Muon:Muon
... MUONHODOPLANE11
... MUONHODOPLANE13
... MUONHODOPLANE12
... MUONHODOPLANE15
... MUONHODOPLANE14
... DUMP
... MUONHODOPLANE16
... SMT
... DUMPIRON
... MAGBOXMBPS2
... MSEL
... MUONHODOSELECTOR
... MBPL
... MBPS2
... MUONHODOPLANE24
... MUONHODOPLANE25
... MUONHODOPLANE26
... DUMPCONCRETE
... MUONHODOPLANE21
... MUONHODOPLANE22
... MUONHODOPLANE23
... MUON10x10A
... MUON10x10C
... MUON10x10B
... MUON10x10D
... MAGBOXMBPL
... SCDUMP
. MYLAREQUIV
. BEAMPIPE2
. S2
. S1
. BEAMPIPE1
. S3
. CALO
.. MuonWall
.. LArCrate
.. Tile:Tile
.. PhantomBarrel
.. CombinedScintillator
. IDET
.. MAGBOXMBPSID
.. MBPSID
... SCT:SCT
... Pixel:Pixel
.. CTBTRT
... TRT:TRT
.. SNEW
athena>
```

ATLAS CTB test beam

Inspection

Terminal — ssh — 52x20

```
athena> SimFlags.SimLayout.print_Flag()
G4AtlasApps.SimFlags:: SimLayout True Rome-Final
athena>
athena> SimFlags.SimLayout.print_Flag()
G4AtlasApps.SimFlags:: SimLayout True Rome-Final
athena> Wv=AtlasG4Eng.G4Eng.Dict.get('volume_world')
athena> Wv.listrecursive_DetFacility()
. IDET
.. SCT:SCT
.. InDetServMat:InDetServMat
.. Pixel:Pixel
.. TRT:TRT
. BeamPipe:BeamPipe
. MUONQ02
.. Muon:Muon
... MSEL
. CALO
.. LArMgr:LArMgr
.. Tile:Tile
athena>
```

ATLAS full simulation

Terminal — ssh — 53x20

```
athena> SimFlags.SimLayout.print_Flag()
G4AtlasApps.SimFlags:: SimLayout True Commissioning-1
athena> Wv=AtlasG4Eng.G4Eng.Dict.get('volume_world')
athena> Wv.listrecursive_DetFacility()
. Atlas
.. IDET
... SCT:SCT
... InDetServMat:InDetServMat
... Pixel:Pixel
... TRT:TRT
.. BeamPipe:BeamPipe
.. TTR_BARREL
.. MUONQ02
... Muon:Muon
.... MSEL
.. CALO
... LArMgr:LArMgr
... Tile:Tile
. CavernInfra:CavernInfra
athena>
```

ATLAS full simulation cosmic rays

For the user the three cases we have described here behaves in the same way.
But if the user does a little "inspection" of what is in the simulation, he can understand what is the simulation he is dealing with

Some use cases(2): Advanced User

“Real case”: a ATLAS CTB test beam user is studying the longitudinal shower profiles in the LAr and he thinks that maybe some material is missing in the “official” simulation. He wants to prove it and for this he wants to add extra material in a far upstream position and also in front of the inner detector.

Here it is how he can customize the simulation (adding the following lines to the standard jobOption)

Terminal — ssh — 82x40

```
#>## this starts the space to customize the simulation #######

from G4AtlasApps import PyG4Atlas

# MATERIAL UPSTREAM
MyMaterialUpstream = PyG4Atlas.DetFacility("ResizeableBox","MyMaterialUpstream")
MyMaterialUpstream.df.SetDx(30.)  # here you play with the dimmensions
MyMaterialUpstream.df.SetDy(50.)
MyMaterialUpstream.df.SetDz(50.)
MyMaterialUpstream.df.SetMaterial("Aluminum")
# here you set the position in mm (careful!, check it with geantino)
MyMaterialUpstream_pos=AtlasG4Eng.G4Eng.gbl.Hep3Vector(-20000.,0.,0.)
MyMaterialUpstream.df.MoveTo(MyMaterialUpstream_pos)
# here you add it to the simulation to the CTB world volume
MyCtb=AtlasG4Eng.G4Eng.Dict_DetFacility.get('CTB')
AtlasG4Eng.G4Eng.add_DetFacility(MyMaterialUpstream,MyCtb)


# MATERIAL IN THE ID

IDmaterial  = PyG4Atlas.DetFacility("ResizeableBox","MID")
IDmaterial.df.SetDx(30.)
IDmaterial.df.SetDy(50.)
IDmaterial.df.SetDz(50.)
IDmaterial.df.SetMaterial("Aluminum")
IDmaterial_position=AtlasG4Eng.G4Eng.gbl.Hep3Vector(-450.,0.,0.)
IDmaterial.df.MoveTo(IDmaterial_position)

# here you add it to the simulation, to the MBPSID volume,
#  this is the reason of the negative coordinates in IDmaterial_position
MyMBPSID=AtlasG4Eng.G4Eng.Dict_DetFacility.get('MBPSID')
AtlasG4Eng.G4Eng.add_DetFacility(IDmaterial,MyMBPSID)


# this is for the check with geantino
G4Command=AtlasG4Eng.G4Eng.gbl.G4Commands()
G4Command.tracking.verbose(1)
#<## this ends the space to customize the simulation #########
                                                  88,0-1        84%
```

Manuel Gallas
CERN PH-SFT

Conclusions:

- PyG4Atlas and G4AtlasApps have proved to be a “flexible”, “quick” and “easy” way to configure simulation applications.

- Different types of users can access, inspect and modify the pre-configured simulation (provided by the G4AtlasApps package) in the same way. Python offers interactivity, and helps to keep job-configuration under control.

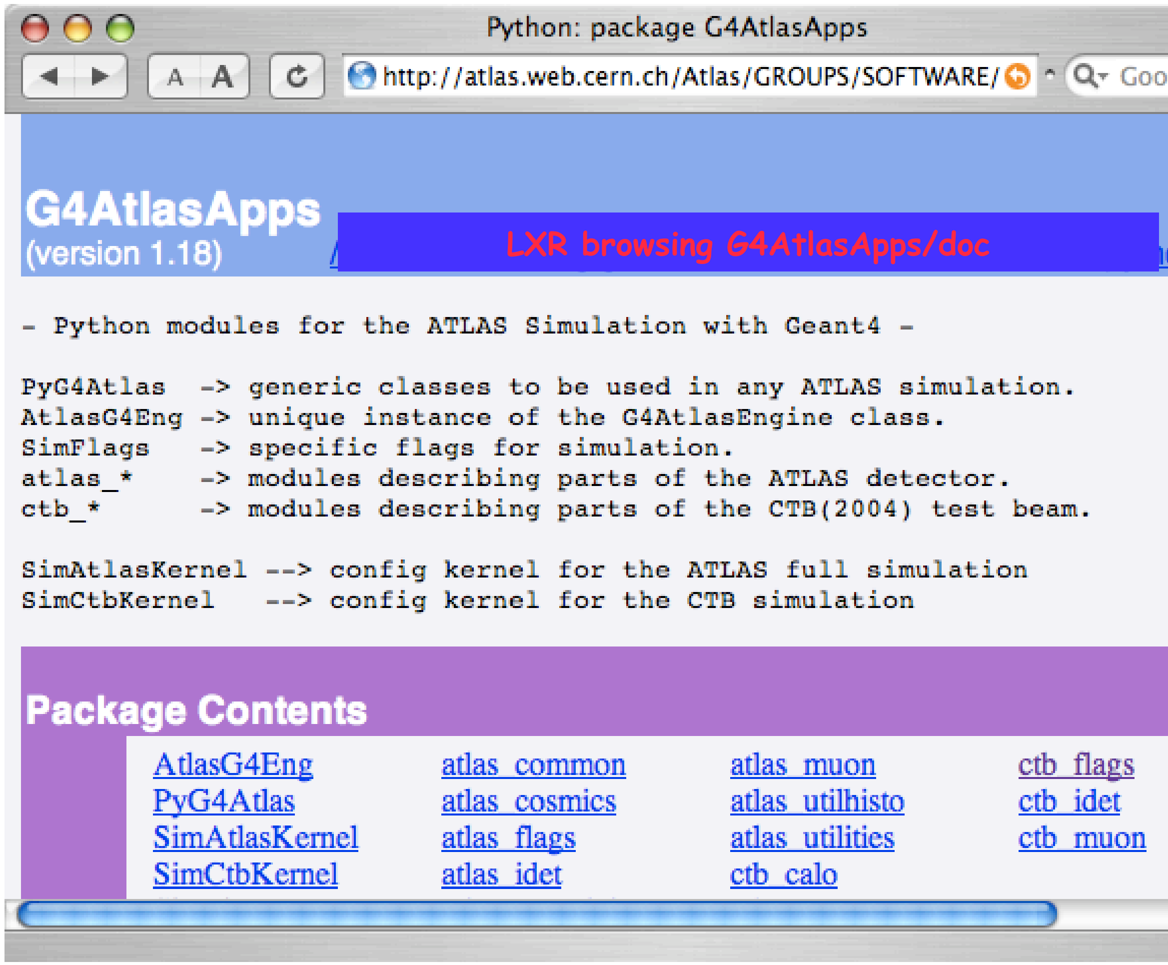
- PyG4Atlas was tested in GRID production for the CTB (ATLAS combined test beam) in which several layouts, many configurations were handled with minimal effort by the production team.

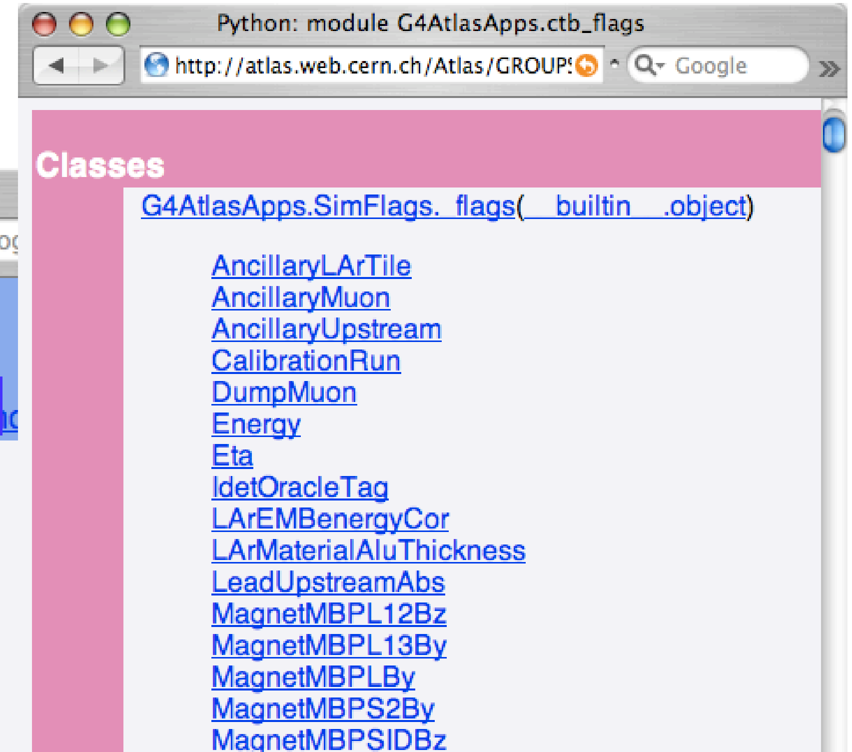
- Geant4+PyLCGDict+Python=PyG4Apps is another possible equation

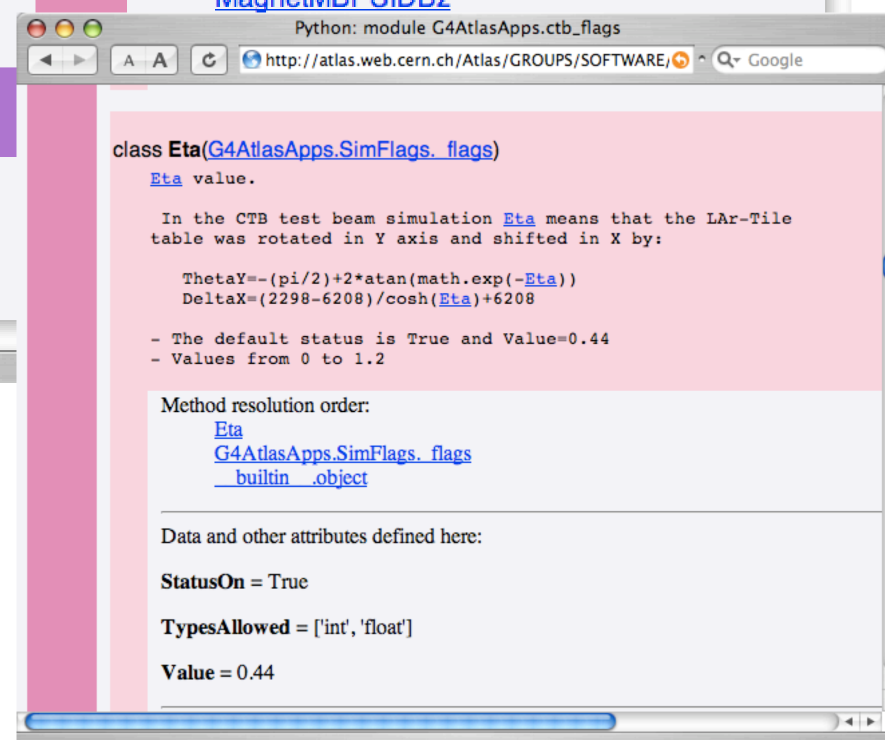
- PyLCGDict is a key point in all the process and not all the classes need to be exported

Backup slides

Backup slides: Flag documentation (I)







Flags and options for simulation (Simflags) are kept updated and full documented in the LXR server.

Backup slides: Flag documentation (II)

Terminal — ssh — 84x39

```
athena> SimFlags.print_Flags()
G4AtlasApps.SimFlags:: AncillaryUpstream True PeriodB ['PeriodA', 'PeriodB', 'Period
C'] ['str']
G4AtlasApps.SimFlags:: MagnetMBPS2By False 0.0 [] ['int', 'float']
G4AtlasApps.SimFlags:: MagnetMBPSIDBz True -1.4 [] ['int', 'float']
G4AtlasApps.SimFlags:: LeadUpstreamAbs False False [] ['bool']
G4AtlasApps.SimFlags:: AncillaryLArTile True True [] ['bool']
G4AtlasApps.SimFlags:: PhysicsList True QGSP_GN ['ExN01', 'ExN02', 'ExN03', 'ExN04',
 'Fast_Physics', 'LHEP_BERT', 'LHEP_GN', 'QGSP_BERT', 'QGSP_GN'] ['str']
G4AtlasApps.SimFlags:: AncillaryMuon True True [] ['bool']
G4AtlasApps.SimFlags:: PersistencyDigit False DigitFile.root [] ['str']
G4AtlasApps.SimFlags:: LArEMBenergyCor True False [] ['bool']
G4AtlasApps.SimFlags:: IdetOracleTag False InnerDetector-CTB-04 ['InnerDetector-CTB-
01', 'InnerDetector-CTB-03', 'InnerDetector-CTB-04', 'InnerDetector-CTB-05', 'InnerD
etector-CTB-06', 'InnerDetector-CTB-08', 'InnerDetector-CTB-09']
G4AtlasApps.SimFlags:: MagnetMBPL13By False 0.0 [] ['int', 'floa
G4AtlasApps.SimFlags:: RunNumber False 0 [] ['int']
G4AtlasApps.SimFlags:: PersistencyHit True ctb_MyOutputFile.root
G4AtlasApps.SimFlags:: MagnetMBPLBy False 0.0 [] ['int', 'float']
G4AtlasApps.SimFlags:: Eta True 0.2 [] ['int', 'float']
G4AtlasApps.SimFlags:: init_Level True 3 [0, 1, 2, 3] ['int']
G4AtlasApps.SimFlags:: ParticleGeneratorOrders True {'t:': ' cons
mX:': ' fixed 1', 'momY:': ' fixed 0', 'vertX:': ' constant -275
xed 0', 'vertY:': ' flat   -10.0 15.0', 'vertZ:': ' flat   -15.0
 constant 50000', 'pdgcode:': ' constant 11'} [] ['dict']
G4AtlasApps.SimFlags:: CalibrationRun False  ['', 'LAr', 'Tile',
G4AtlasApps.SimFlags:: KinematicsMode False SingleParticle ['Sing
Generator', 'ReadGeneratedEvents'] []
G4AtlasApps.SimFlags:: SimLayout True ctbh8_combined ['ctbh8_comb
n', 'ctbh8_lar-material'] ['str']
G4AtlasApps.SimFlags:: MagnetMBPL12Bz False 0.0 [] ['int', 'floa
G4AtlasApps.SimFlags:: Energy True 50000 [] ['int', 'float']
G4AtlasApps.SimFlags:: RunConditions False  [] ['str']
G4AtlasApps.SimFlags:: ParticlePDG True 11 [] ['str']
G4AtlasApps.SimFlags:: LArMaterialAluThickness False 0.0 [] ['fl
G4AtlasApps.SimFlags:: Seeds True SINGLE 2000160768 643921183 []
G4AtlasApps.SimFlags:: DumpMuon False False [] ['bool']
G4AtlasApps.SimFlags:: EvgenInput False  [] ['str']
athena>
```

Ask for help!!

athena> **help(SimFlags.IdetOracleTag)**

Terminal — ssh — 81x39

```
Help on class IdetOracleTag in module G4AtlasApps.ctb_flags:

class IdetOracleTag(G4AtlasApps.SimFlags._flags)
 |  Oracle tag for the Inner detector geometries.
 |
 |  values: 'InnerDetector-CTB-01/03/04/05/06/08/09'
 |
 |  Method resolution order:
 |      IdetOracleTag
 |      G4AtlasApps.SimFlags._flags
 |      __builtin__.object
 |
 |  Data and other attributes defined here:
 |
 |  TypesAllowed = ['str']
 |
 |  Value = 'InnerDetector-CTB-04'
 |
 |  ValuesAllowed = ['InnerDetector-CTB-01', 'InnerDetector-CTB-03', 'Inne...
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from G4AtlasApps.SimFlags._flags:
 |
 |  __init__(name)
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from G4AtlasApps.SimFlags._flags:
 |
 |  print_Flag(self) from __builtin__.type
 |
 |  print_FlagFull(self) from __builtin__.type
 |
 |  set_Off(self) from __builtin__.type
 |
 |  set_On(self) from __builtin__.type
 |
 |  set_Value(self, n_Value) from __builtin__.type
 |
:
```

Flags and options for simulation (Simflags) are written in the log files and can be accessed in interactive mode, (help, meaning, possible values etc)