Journal of Physics: Conference Series **219** (2010) 032001 doi:10.1088/1742-6596/219/3/032001

Monte Carlo Generators in ATLAS software

C.Ay[1], A. Buckley[2], J. Butterworth[3], J. Ferland[4], I. Hinchliffe[5], O. Jinnouchi[6], J. Katzy[7], B. Kersevan[8], E. Lobodzinska[7], J. Monk[9], Z. Qin[7], V. Savinov[10], J. Schumacher[11]

[1] University Goettingen, II physics institutes of Physics, Germany
[2] Institute for Particle Physics Phenomenology, Durham University, UK
[3] Dept. of Physics and Astronomy, University College London
[4] University of Montreal, Montreal
[5] Lawrence Berkeley National Laboratory, Berkeley, CA , 94720 USA
[6] KEK, High Energy Accelerator Research Organization, 1-1 Oho, Tsukuba 305-0801, Japan
[7] DESY, Hamburg, Germany
[8] Faculty of Mathematics and Physics, University of Ljubljana, Jadranska 19, SI-1000 Ljubljana, Slovenia
[9] University College London, Department of Physics and Astronomy, Gower Street, London WC1E6BT, United Kingdom
[10] University of Pittsburgh, Dept. of Physics and Astronomy 3941 O'Hara Street, Pittsburgh, PA 15260, USA
[11] TU Dresden, Institute fuer Kern- und Teilchenphysik, D-01069 Dresden, GERMANY

Abstract.
This document describes how Monte Carlo (MC) generators can be used in the ATLAS software framework (Athena). The framework is written in C++ using Python scripts for job configuration. Monte Carlo generators that provide the four-vectors describing the results of LHC collisions are written in general by third parties and are not part of Athena. These libraries are linked from the LCG Generator Services (GENSER) distribution. Generators are run from within Athena and the generated event output is put into a transient store, in HepMC format, using StoreGate. A common interface, implemented via inheritance of a GeneratorModule class, guarantees common functionality for the basic generation steps. The generator information can be accessed and manipulated by helper packages like TruthHelper. The ATLAS detector simulation as well access the truth information from StoreGate[1]. Steering is done through specific interfaces to allow for flexible configuration using ATLAS Python scripts. Interfaces to most general purpose generators, including: Pythia6, Pythia8, Herwig, Herwig++ and Sherpa are provided, as well as to more specialized packages, for example Phojet and Cascade. A second type of interface exist for the so called Matrix Element generators that only generate the particles produced in the hard scattering process and write events in the Les Houches event format. A generic interface to pass these events to Pythia6 and Herwig for parton showering and hadronisation has been written.

1. Introduction

The ATLAS (**A T**oroidal **LHC A**pparatu**S**) experiment [1, 2], one of the LHC experiments [3, 4], has a rich physics programme, which includes precise measurements of known Standard Model

[1] StoreGate is a toolkit that allows client algorithms to interact with the transient data store to record or retrieve data objects.

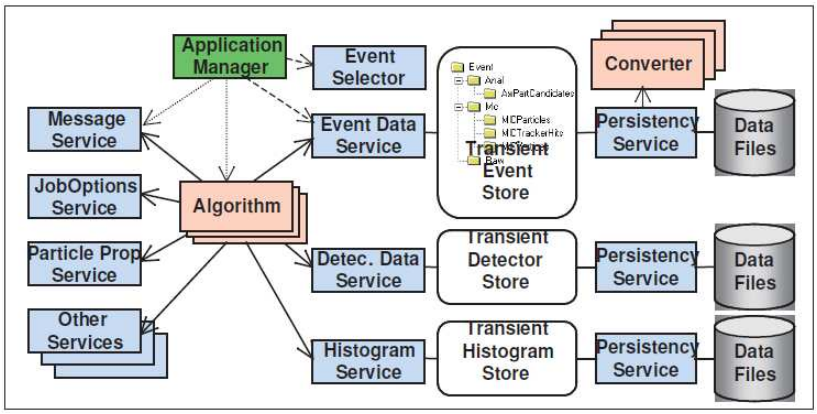

Figure 1. Object Diagram of the GAUDI Architecture [6].

(SM) processes like top-physics, electroweak-physics and the search for the Higgs boson. In addition the experiments search for physics beyond the SM, e.g. new physics like supersymmetry (SUSY) and other extensions of the SM.
Due to the huge background from QCD processes, multiple interactions and underlying events a detailed simulation of signal and background processes is required. The definition of background processes is not strict. Ones signal may be the background for others. To take the whole spectra of physics analyses into account the existing multi-purpose and special-purpose generators need to be included or interfaced to the ATLAS framework. The interface needs to provide the possiblity to steer the generator parameters to allow for tuning of the Monte Carlo data and variation of existing models.
This document will first give an introduction to the ATLAS software framework, followed by a description on how external generators are interfaced into the framework, depending on their type. Finally it presents the configuration possibility, including examples, to generate events locally or in a distributed computing system.

2. The ATLAS framework Athena

The ATLAS framework is called Athena [5]. Athena is a control framework based on the GAUDI [6, 7] component architecture originally developed by LHCb [8]. The Gaudi Architecture Object Diagram is shown in Figure 1. It consists of an application manager, services and algorithms. Services refer to classes whose job is to provide a set of facilities or utilities to be used by other components, e.g. algorithms. The functionality of an algorithm is to take input data, manipulate it and produce new output data.

The algorithms included in Athena are almost exclusively written in C++. The framework model produces only one executable application. The other components produce shared libraries, which typically contain algorithms for data processing, or services for the algorithms. Scripts for run time control of code execution are written in Python. The algorithms are configured and sequenced at run time using job options [9] interpreted by the core software. Three steps are performed in a standard Athena job:

- **Initialization**
  Services and Algorithms are loaded on demand
- **Event Loop**
  Algorithms in list run sequentially on each event
- **Finalization**
  Algorithms are terminated and objects are deleted.

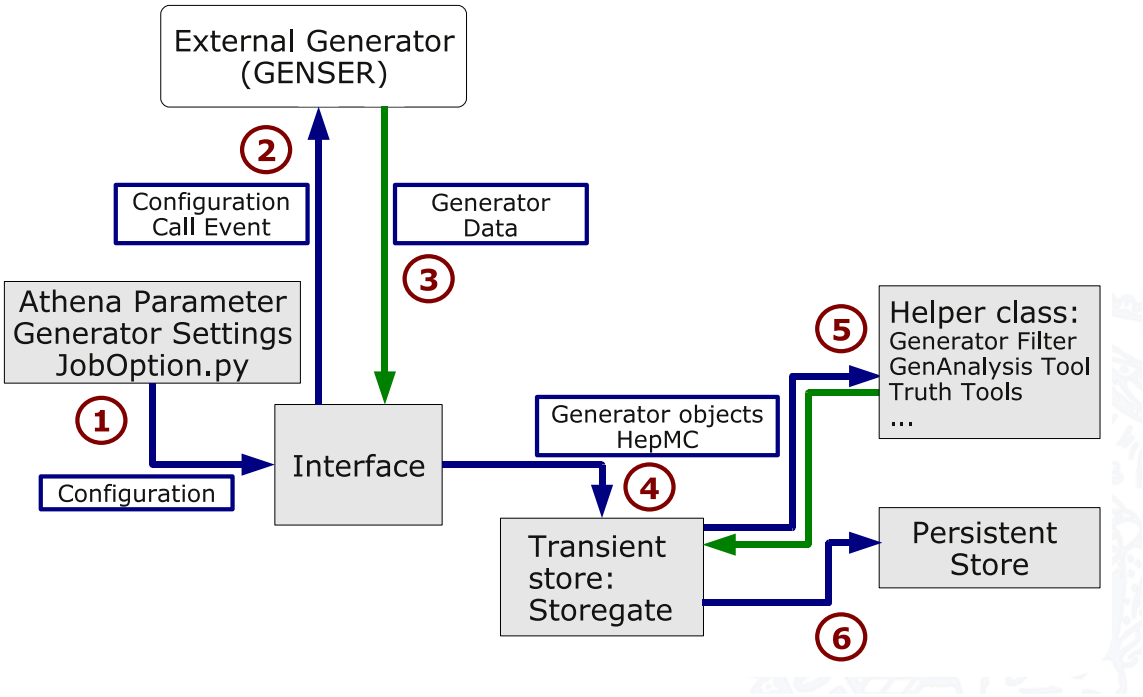

Figure 2. Default workflow for generators in Athena.

3. Integration of external generators in Athena

The Monte Carlo (MC) generator codes are developed independent of the experiments. To unify the access to the generator libraries the Generator Services project [10] (GENSER) collaborates with the LHC experiments. GENSER provides validated LCG [11] compliant code for both the theoretical and experimental communities at the LHC.

In Athena the Configuration Management Tool [12] (CMT) provides a standard formalism to make these generator libraries available to be used in the framework. Those light-weight packages are called "glue"-packages and mainly consist of one requirements file only. In addition an interface package is needed to provide the steering possibility and to actually call the generator routines. More details about the interface package mechanism are reported in Section 3.1.

With these two extra packages per generator a default workflow for a generator job in Athena can be drawn, as illustrated in Figure 2:

1. The Athena parameter and generator specific settings are passed to the interface package via the job options,
2. During the initialization step the generator will be configured by the interface according to the settings in the job options. In the event loop the interface calls the generator and the event will be generated.
3. The generator returns the generated event back to the interface,
4. The interface stores the generated event in HepMC format [13] in the transient store named Storegate,
5. Helper classes, e.g. GeneratorFilter[2] and TruthTools[3], get the HepMC container from the transient store, modify[4] the event according to the selected algorithms and write the

[2] A package for generator event filtering in Athena.
[3] A package for selecting particles on MC Truth level in Athena.
[4] The event can be removed. Event information can be altered, removed and/or added.

modified event back into Storegate. The event can also be skipped. Simulation and other downstream processing steps will also access the HepMC container via StoreGate,

6. The events can be read out of Storegate, converted to a root-style format and written out to a persistent store, namely an output file.

Depending on the generator the steps 2 and 3 are realized in different ways in the interface packages.

3.1. Generator Interface Packages

The interface packages inherit from the common base class, GenModule. This one provides the basic functions, which need to be customized for each generator:

- `initialize()`: Configuration of the generator,
- `execute()`: Call generator routines to generate event, convert event to HepMC format and write event into Storegate,
- `finalize()`: Call destructors

Usually the interface configuration options are closely linked to the configuration parameters of the generator to be interfaced. The name of the generator interface packages is constructed from the name of the generator to be interfaced plus the string `_i`, e.g. `Pythia_i`, `Herwig_i` and `Madgraph_i`.

3.2. Athena Organization and Management

The software is large both in size and also in the number of developers involved. Therefore policies and technical methods to make the development and release mechanisms as smooth and efficient as possible, have been put in place. Some of these mechanism will be presented briefly in this section. A more detailed description is given in [14].
The ATLAS software is physically organized using the well-known Concurrent Versions System [15] (CVS) for version control. The repository is organized in a directory structure, with each developer group owning a top-level directory, and with the actual code (packages) contained in sub-directories.
To formalize software production CMT is used, which generates appropriate commands to build the software package based on information supplied by the developer in a requirements file. CMT is used for release building.
The Tag Collector [16] is used for collecting the packages that build a release. In addition there is NICOS (**NI**ghtly **CO**ntrol **S**ystem) [17], which runs the nightly builds of the offline software.

4. Generators for the ATLAS experiment

A long list of generators[5] are used in Athena and te addition of more generators is foreseen. All these generators can be grouped in three categories.
The first group of generators, the full generators, include parton shower and fragmentation. Generators belonging to that group are Pythia [19], Herwig [20], Sherpa [21], Hijing[6], Pythia8 [22] and Herwig++ [23].
Specific purpose add-on packages to generators like Tauola [24], Phojet [25]and Photos [24] represent the second group. For those packages the workflow differs slightly from the default workflow described in Section 3. As show in Figure 3 the add-on packages retrieve the HepMC container from Storegate, modify the events and finally write the HepMC events back into Storegate.

[5] currently about 30, [18]
[6] Hijing uses some Pythia Tools

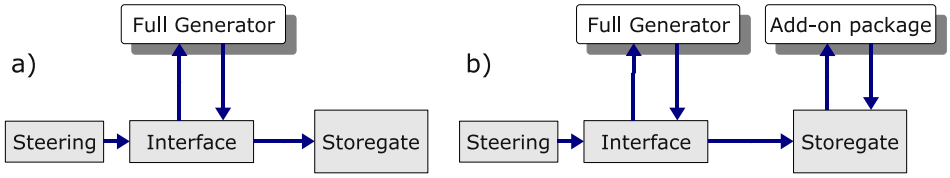

Figure 3. workflow fragment for a) full generator and b) for add-on package.

Most of the generators belong to the third group, the parton level generators, which require a full generator like Pythia and Herwig to perform the parton shower and the fragmentation. For that purpose the full generators provide a common format, the LHA (**L**es **H**ouches **A**ccord) format, alternatively the LHEF (**L**es **H**ouches **E**vent **F**ile) format. The LHA format is a well defined standard and consists of the FORTRAN common blocks HEPRUP and HEPUP, which only reside in Memory. The LHA format is described in detail in [26] and used by AcerMC [27], Cascade [28] and others. For those kind of generators the interface first need to call the parton level generator and then provide the events in LHA format to the relevant full generator. The full generator than returns the events in HepMC format back to the interface.
The LHEF format describes a dump of the LHA Fortran Common blocks into a file. The "old style LHEF" used to be a generator specific non-standard ASCII-dump. Therefore it was agreed to use a standardized XML dump of the LHA Common Block described in [29]. This format is well-defined and gives also the possibility to add comments, headers and other extensibles. This is the preferred format by ATLAS. Among others the generators MadGraph [30, 31], Baur [32], Winhac[7] [33] and MC@NLO[8] [34] use the LHEF format. Those kind of generators need to generate the events in advance. The LHEF file can than be interpreted by the interface, converted to LHA format and passed to the full generator.
Generators that need to be integrated are requested to provide their events in the new LHEF format and use the generic interface `LHEF_i`[9] to pass their events to Athena.

In Appendix A an easy example is shown on how to generate 5 Pythia events and print out the HepMC output. This example basically can be used to generate also other events, if the appropriate interface package is chosen.

5. Transformations

Generating Monte Carlo events for the supported generators with Athena on production scale requires more flexible objects than the job options as described in Appendix A.
The ATLAS software framework provides dedicated Python scripts, called transformations, that take a skeleton Athena job option Python file as an input and a set of command-line parameters that allow the user to run an arbitrary number of Athena jobs based on that skeleton job option Python file. The ATLAS Monte Carlo generation exploits Athena's transformation mechanism providing readily available transformations, that allow users to run generation via simple command-line options. For example using the basic event generation transformation the user can select the type of event to generate, how many events and one of the available Monte Carlo generator to produce them. These Transformations provide a convenient and flexible mechanism to use predefined job options fragments, which can be controlled via command line arguments and/or grid tools. The advantages are

[7] Baur and Winhac use "old style LHEF" with conversion script to new LHEF format
[8] "old style LHEF"
[9] this package does not follow the naming scheme for generator interface packages

- one job transform Python script to generate events for any generator integrated in Athena locally or on the Grid according to the given arguments,
- commonly used parameters in the job option fragments, which can be selected by the arguments are centrally controlled and maintained,
- the generatted data is reproducible.

The components and a example transformation script is shown in appendix Appendix B. Three differen kinds of transformations are available:

1. event generation: `csc_evgen_trf.py`
2. related to simulation: `csc_atlasG4_trf.py`, `csc_digi_trf.py`, `csc_simul_trf.py`
3. related to reconstruction: `csc_recoESD_trf.py`, `csc_recoAOD_trf.py`, `csc_reco_trf.py`

A detailed description is given in [35].
Grid tools like Ganga [36] and Pathena [37] support the transformation mechanism.

6. Summary

The ATLAS software Athena is a control framework that is written in C++ and can be controlled via Python scripts called job options. The external generators provided by GENSER are interfaced to the framework using specially designed interface packages and can be grouped in three categories: full generators, specific purpose add-on packages and parton level generators. Most of the generators used in Athena belong to the third group and require an additional full generator to perform the fragmentation and hadronisation. The parton level generator parse the generated events via LHA or LHEF to the full generator. Instead of simple and static job options, more complex and flexible job Transformations are used for production scale Monte Carlo event generation.

Appendix A. Example job options file

To use the ATLAS software a setup of CMT and Athena is required. A detailed description on how to use Athena is given in the workbook in [38].
The following job options file (jobOptions.py) will generate 5 Pythia events and print out the HepMC output:

```
from AthenaCommon.AppMgr import ServiceMgr
ServiceMgr.MessageSvc.OutputLevel = DEBUG

theApp.EvtMax = 5

from AthenaServices.AthenaServicesConf import AtRndmGenSvc
ServiceMgr += AtRndmGenSvc()
ServiceMgr.AtRndmGenSvc.Seeds = ["PYTHIA 4789899 989240512",
 "PYTHIA_INIT 820021 2347532"]

from AthenaCommon.AlgSequence import AlgSequence
job=AlgSequence()

from Pythia_i.Pythia_iConf import Pythia
job +=Pythia()
job.Pythia.PythiaCommand = ["pysubs msel 13","pysubs ckin 3 18.",
    "pypars mstp 43 2"]
job.Pythia.PythiaCommand += ["pypars mstp 51 19070", "pypars mstp 52 2",
```

```
                                "pypars mstp 53 19070", "pypars mstp 54 2",
                                "pypars mstp 55 19070", "pypars mstp 56 2"]

from TruthExamples.TruthExamplesConf import DumpMC
job += DumpMC()
```

To run the code execute the command

```
athena jobOptions.py
```

The Application Manager `AppMgr` (see Figure 1) is the main instance and controls the Service Manager `ServiceMgr` and algorithms. In this example two services are shown, the Message Service `MessageSvc` (see Figure 1) and the Randomnumber Generator Service `AtRndmGenSvc`, which are controlled by `ServiceMgr`. In this example the Randomnumber Generator Service provides the random numbers for the generator Pythia using the specified seeds. In this case different seeds are taken for the initialization and the event generation.
To run a specific algorithm an AlgSequence `job` need to be defined and the algorithms need to be added to the sequence. In this example the class `Pythia` defined in the interface package `Pythia_i` and the class `DumpMC` defined in the package `TruthExamples` are listed in the sequence and will be executed in a sequence for every event.

Appendix B. Components of Transformations

The Python executable script defining transformation are usually located in the `scripts` directory in the appropriate package (e.g. `EvgenJobTransforms`) and have the ending `_trf.py`. A minimal job transformation script looks like this:

```
from PyJobTransformsCore.trf import JobTransform
from PyJobTransformsCore.full_trfarg import *
from MyPackage.MyConfig import myConfig

class MyTransform(JobTransform):
def __init__(self):
JobTransform.__init__(self, skeleton='skeleton.test.py', config=myConfig )
self.add ( MaxEvents() )

# execute it if run as a script
if __name__ == '__main__':
# make transform object
trf = MyTransform()

import sys
sys.exit( trf.exeSysArgs().exitCode() )
```

The following components are needed:

- A skeleton job options file used to run Athena. The file is usually located in the `share` directory and follow the naming scheme `skeleton.*.py` (skeleton.test.py).
- A run argument job options file, created on-the-fly at runtime to make the command line arguments available to the skeleton.
- A configuration object (a Python object) to steer the skeleton job options file (make it configurable).

References

[1] 1999 *ATLAS detector and physics performance: Technical Design Report, 2* Technical Design Report ATLAS (Geneva: CERN) electronic version not available
[2] Http://atlas.ch/
[3] Pinfold J L 2004
[4] Http://public.web.cern.ch/PUBLIC/en/LHC/LHCExperiments-en.html
[5] Https://twiki.cern.ch/twiki/bin/view/Atlas/AthenaFramework
[6] Barrand *et al* G 2000
[7] Http://cern.ch/proj-gaudi/
[8] Http://lhcb.web.cern.ch/lhcb/
[9] Https://twiki.cern.ch/twiki/bin/view/Atlas/JobOptions
[10] Http://lcgapp.cern.ch/project/simu/generator
[11] 2002 The lhc computing grid project : Lcg. oai:cds.cern.ch:541137 Tech. rep. CERN Geneva cERN, Geneva, 11, 12, 13, 14, 15 Mar 2002
[12] Http://www.cmtsite.org/
[13] Http://lcgapp.cern.ch/project/simu/HepMC/
[14] Obreshkov *et al* E 2006 Organization and management of atlas offline software releases Tech. Rep. ATL-SOFT-PUB-2006-008. CERN-ATL-SOFT-PUB-2006-008. ATL-COM-SOFT-2006-013 CERN Geneva
[15] Http://www.nongnu.org/cvs/
[16] Http://atlastagcollector.in2p3.fr
[17] Http://www.usatlas.bnl.gov/computing/software/nicos/
[18] Https://twiki.cern.ch/twiki/bin/view/AtlasProtected/McGeneratorsForAtlas
[19] Sjstrand T, Mrenna S and Skands P Z 2006 *J. High Energy Phys.* **05** 026. 570 p
[20] Corcella G, Knowles I G, Marchesini G, Moretti S, Odagiri K, Richardson P, Seymour M H and Webber B R 2002 Herwig 6.5 release note. herwig 6.5 Tech. Rep. hep-ph/0210213. CAVENDISH-HEP-2002-17. CERN-TH-2002-270. DAMTP-2002-124. IPPP-2002-58 CERN Geneva
[21] Gleisberg T and Krauss F 2007 *Eur. Phys. J. C* **53** 501–523. 42 p comments: 42 pages, 8 figures
[22] Sjstrand T, Mrenna S and Skands P 2007 *Comput. Phys. Commun.* **178** 852867. 27 p
[23] Gieseke S, Grellscheid D, Hamilton K, Ribon A, Richardson P, Seymour M H, Stephens P and Webber B R 2006 Herwig++ 2.0 release note. oai:cds.cern.ch:986755. herwig++ 2.0 Tech. Rep. hep-ph/0609306. CERN-PH-TH-2006-182. CAVENDISH-HEP-2006-18. IFJPAN-IV-2006-6 CERN Geneva
[24] Golonka P, Kersevan B P, Pierzchala T, Richter-Was E, Was Z and Worek M 2003 *Comput. Phys. Commun.* **174** 818–35
[25] Engel R 1994 *Z. Phys. C* **66** 203–214. 26 p
[26] Boos *et al* E 2001 Generic user process interface for event generators Tech. Rep. hep-ph/0109068
[27] Kersevan B P and Richter-Was E 2002 The monte carlo event generator acermc 1.0 with interfaces to pythia 6.2 and herwig 6.3 Tech. Rep. hep-ph/0201302. ICSJU-2002-001. TPJU-2002-1 Cracow Univ. Dept. Theor. Phys. Cracow
[28] Jung H and Salam G P 2000 *Eur. Phys. J. C* **19** 351–360. 20 p
[29] Alwall *et al* J 2006 *Comput. Phys. Commun.* **176** 300–304. 8 p
[30] Alwall J, Demin P, de Visscher S, Frederix R, Herquet M, Maltoni F, Plehn T, Rainwater D L and Stelzer T 2007 *J. High Energy Phys.* **09** 028. 38 p
[31] Stelzer T and Long W F 1994 21 p
[32] Baur U, Han T and Ohnemus J 1993 *Phys. Rev. D* **48** 5140–5161. 52 p
[33] Placzek W and Jadach S 2003 *Eur. Phys. J.* **C29** 325–339 (*Preprint* `hep-ph/0302065`)
[34] Frixione S and Webber B R 2002 *J. High Energy Phys.* **06** 029. 69 p
[35] Https://twiki.cern.ch/twiki/bin/view/Atlas/AtlasPythonTrf
[36] Brochu *et al* F 2009 Ganga: a tool for computational-task management and easy access to grid resources Tech. Rep. arXiv:0902.2685
[37] Http://www-hep2.fzu.cz/twiki/bin/view/ATLAS/Pathena
[38] Https://twiki.cern.ch/twiki/bin/view/Atlas/WorkBook