Table of Contents

6.1 Thirty-Minute Introduction to Generation and Simulation

Complete:
Detailed Review status

Goals of this page

This page provides an entry point to the physics event generation and detector simulation in CMSSW.
It should serve as a jump-start for people who will need to prepare configuration application for central production of the Monte Carlo samples, as representatives of their Physics groups.
Another group of people we aim onto are those who will need to produce their Monte Carlo samples on a private basis, i.e. outside of central production.

Contents

Introduction

Physics event generation and detector simulation are the earliest steps in the event processing chain that leads to producing a Monte Carlo samples suitable for physics analysis.
An ample variety of the Monte Carlo samples for various types of physics analysis are **produced and distributed centrally**. They can be found via the DAS page. However, if you can not find the event sample you would like to use (or compatible with the CMSSW version that you are using), you may want to produce your own Monte Carlo samples ("private samples").

In this section we will present several How-To's on the most often used features and utilities, will show several examples, and will provide links to more detailed documents.

In general, a user should assume **CMSSW_10_2_X and higher** release cycle. Where necessary, we'll provide specific tips, especially for most recent releases.
At the same time, we would like to mention that newer developments have happened in more recently released, and imposed modifications of certain details. We will provide specific comments and suggestions when applicable.

Generation of high-energy-physics events, i.e. sets of outgoing particles produced in the interactions between two incoming particles, **must** be the first step in the Monte Carlo event processing chain.
In CMSSW we have interfaces to many physics event generators that are of interest to the collaboration. In this writeup, we will present an example that will use Pythia8 event generator, as it has been most heavily used in production so far.

The following step(s) can be either:

- Simulation and Digitization, whereby the newly generated particles are run through detailed, Geant4-based simulation of the CMS detector, and detector electronics response is modelled. After this step, you need more (reconstruction, AOD and possibly MINIAOD) in order to get samples actually comparable with data.
- Fast Simulation, which uses a parametric approach to simulate and reconstruct events with the CMS detector; the concept of FastSim is to reduce the CPU time overhead, while still benefiting from an accurate simulation of the detector effects, in view of doing physics analysis, develop and tune reconstruction algorithms, design detector upgrades, etc. This step produces files already comparable with data.

CMSSW offers a large collection of software tools, utilities, scripts, and pre-fabricated configuration application fragments. Thus, in most cases, a user will only need to know how to find the right components, to compose them together, and to execute - this will be the focus of this section.

The tips we offer here have been tested on the LXPLUS cluster at CERN. We have also tested on remote sites as well and will provide remarks and additional guidance as needed.

Running generation and simulation in CMSSW

Running the event generation and simulation in CMSSW means running a framework job with the usual syntax:

```
cmsRun <My configuration file>
```

Here `cmsRun` is the principal CMSSW executable that gets configured to do one or another type of job by the input <*My configuration file*>, where a user specifies desired CMSSW software components and the order of their execution.

In order to "find" `cmsRun` and to be able to run it, you will need to have it in your PATH, which will be done by setting up your run time environment. A quick sketch of doing so is given below:

```
cmsrel CMSSW_X_Y_Z
cd CMSSW_X_Y_Z/src
cmsenv
```

(of course, in this sketch X_Y_Z is a "generic" form that means one or another CMSSW release).

When it comes to the configuration input file, one needs to remember that a step in the event processing chain may create event data (`edm::Event`) that will serve as an input to the subsequent steps, thus it is important to properly order the sequence of steps for execution. One also need to know that some software components may need other "helper" software, thus those services and modules also need to be present in the configuration file.
However, most users will not need to worry about such details as in CMSSW there are utilities which will ensure that all service software is included, and the sequence of steps in the event processing chain is correct.

We would like to offer an example configuration card (rename to .py only), which is similar to those used in central production and can be directly executed with `cmsRun` in CMSSW_5_3_26. It configures `cmsRun` to generate an RS graviton G(1 TeV) decaying to ZZ. Then it processes the Higgs events through Geant4-based detector simulation, digitization, reconstruction etc. At the end of the job, it writes out an output file. You can copy these configuration files into your work area and execute `cmsRun` on it.
Below we will provide tips on how you can compose your own configuration application, for both Full or Fast Simulation.

Composing Full Simulation Configuration Application

Later in this document, we will offer a quick walk-through the example configuration card, and will point your attention to details that are specific to event generation and simulation.

Here we would like to stress that this configuration application has been composed via **standard** CMSSW utility called `cmsDriver.py`. This utility will appear in your PATH once you setup your run time CMSSW environment (just like `cmsRun`). More detailed information on `cmsDriver.py` can be found in the cmsDriver documentation.

In this document we will offer **quick tips** on the `cmsDriver.py` usage. You can re-create this configuration application by executing the following commands:

To use fragments from the repository one can browse the repository on the web and copy or download the fragment with curl, e.g.

```
curl -s https://github.com/cms-sw/genproductions/blob/master/python/ThirteenTeV/RSGraviton/RSGrav

scram b
```

Otherwise, you can create a new empty directory and clone the full repository

```
mkdir -p Configuration/GenProduction/
git clone git@github.com:cms-sw/genproductions.git Configuration/GenProduction/
cd Configuration/GenProduction/python/ThirteenTeV/RSGraviton/
scram b

CMSSW_7_0_X

cmsDriver.py
Configuration/GenProduction/python/ThirteenTeV/RSGraviton/RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8
--fileout file:RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_FULLSIM.root --mc
--eventcontent AODSIM --datatier GEN-SIM --conditions auto:mc --beamspot
Realistic8TeVCollision --step GEN,SIM,DIGI,L1,DIGI2RAW,HLT:GRun,RAW2DIGI,L1Reco,RECO
--python_filename RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_FULLSIM_cfg.py
--no_exec --customise Configuration/DataProcessing/Utils.addMonitoring -n 64

CMSSW_9_2_X

cmsDriver.py
Configuration/GenProduction/python/ThirteenTeV/RSGraviton/RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8
--fileout file:RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_GEN-SIM.root -s
GEN,SIM --mc --datatier GEN-SIM --beamspot Realistic25ns13TeVEarly2017Collision
--conditions auto:phase1_2017_realistic --eventcontent RAWSIM --era Run2_2017
--python_filename RSGravitonToZZ_kMpl01_M_1000_pythia8_GEN-SIM_Run2_2017_cfg.py --no_exec
-n 50

CMSSW_9_3_X

cmsDriver.py
Configuration/GenProduction/python/ThirteenTeV/RSGraviton/RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8
--fileout file:RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_GEN-SIM.root -s
GEN,SIM --mc --datatier GEN-SIM --beamspot Realistic25ns13TeVEarly2017Collision
--conditions auto:phase1_2017_realistic --eventcontent RAWSIM --era Run2_2017
--python_filename RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_GEN-SIM_cfg.py
--no_exec -n 50

step1:DIGI2RAW-HLT (DR)
```

```
cmsDriver.py -s DIGI,L1,DIGI2RAW,HLT --datatier GEN-SIM-RAW --conditions
auto:phase1_2017_realistic --eventcontent RAWSIM --era Run2_2017 --filein
file:RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_GEN-SIM.root --fileout
file:RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_GEN-SIM-RAW.root
--python_filename
RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_FULLSIM_GEN_SIM_RAW_cfg.py -n -1
--no_exec

step2:RECO

cmsDriver.py -s RAW2DIGI,L1Reco,RECO --datatier RECO --conditions
auto:phase1_2017_realistic --eventcontent AODSIM --era Run2_2017 --filein
file:RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_GEN-SIM-RAW.root --fileout
file:RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_GEN-SIM-RAW-RECO.root
--python_filename
RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_FULLSIM_GEN_SIM_RAW_RECO_cfg.py -n
-1 --no_exec

For Run2
 step0 : GEN-SIM

setenv SCRAM_ARCH slc7_amd64_gcc700

source /cvmfs/cms.cern.ch/cmsset_default.csh

scram p CMSSW CMSSW_10_2_15_patch1

cd CMSSW_10_2_15_patch1/src

eval `scram runtime -csh`

curl -s --insecure https://cms-pdmv.cern.ch/mcm/public/restapi/requests/get_fragment/BPH-RunIIFal
[ -s Configuration/GenProduction/python/BPH-RunIIFall18GS-00170-fragment.py ]

scram b

cmsDriver.py Configuration/GenProduction/python/BPH-RunIIFall18GS-00170-fragment.py --fileout fil

cmsRun BPH-RunIIFall18GS_cfg.py


 step1:DIGI2RAW-HLT (DR)

voms-proxy-init

cmsDriver.py step1 --filein file:BPH-RunIIFall18GS.root --fileout file:BPH-RunIIAutumn18DR_step1.

cmsRun BPH-RunIIAutumn18DR_cfg.py


 step2: RECO

cmsDriver.py step2 --filein file:BPH-RunIIAutumn18DR_step1.root --fileout file:BPH-RunIIAutumn18D

cmsRun -e -j BPH-RunIIAutumn18DR_2_rt.xml BPH-RunIIAutumn18DR_2_cfg.py


 Step3: MiniAODSIM

cmsDriver.py step1 --filein file:BPH-RunIIAutumn18DR_step2.root --fileout file:BPH-RunIIAutumn18M
```

curl (like wget) just fetches an input file from a web location (in this case we used this fragment) and places it in a "fake" CMS package creating a directory structure like those of the CMS packages. NOTE THAT Configuration/GenProduction IS NOT A PART OF CMSSW.

Now let us briefly review the input arguments to `cmsDriver.py` utility.

- The first - and mandatory - input argument to `cmsDriver.py` is configuration fragment that determines what physics event generator you wish to use and what topology you intend to generate. In this example we use a pre-fabricated fragment which is originally located in the genproductions area. For the majority of applications for producing Monte Carlo samples the **only difference will be this generator-level configuration fragment**, while other conditions and steps will be **standard**. This is a great benefit of using `cmsDriver.py` for composing applications for Monte Carlo production, as it will ensure that most current setups and conditions will be employed.
- The `-s` field contains the sequence of event processing steps. In the above example the chain starts with the GEN(eration), including necessary filters to select events of users specific interest where applicable, following with SIM(ulation), DIGI(tization), L1 trigger emulation, conversion of the simulated DIGI2RAW (raw data format), and H(igh)L(evel)T(triggers) simulation and RECO(nstruction). The last steps are not a part of the event generation and simulation domain, but this is how the event processing chain in composed in production of the Monte Carlo samples. The last steps are also performed on real data. If you wish to terminate your chain at the generation level, you may use "-s GEN". However, if you intend to run "private production" we suggest that you compose the chain up to the last step.
- Details of the `--conditions` field can be found in the Software Guide on the FrontierConditions. Notice that the choice `--conditions auto:mc` chooses the best conditions for a given release.
- To select the content of the `--datatier` field plese view the documentation on the allowed Data Tiers.
- Content of the `--eventcontent` field is described in great details in the SWGuideDataFormatTable.
- In the `-n` field you will specify how many events you want to generate, simulate, etc.
- The `--no_exec` argument tells `cmsDriver.py` to write out the configuration file. If you do not specify this argument, `cmsDriver.py` will proceed to executing `cmsRun`.
- The `--python_filename` and `--fileout` define the output configuration file and the output root file after `cmsRun`, respectively
- The customization part is not important for private production and may be skipped

Composing Fast Simulation Configuration Application

From the Generator point of view, Fast Simulation is identical to the Full Simulation both in usage of single-particle gun and one of the multi-purpose event generators as described above.

However, when using `cmsDriver.py` command for Fast Simulation there are some differences compared to the Full Simulation case described above:

- First of all, Fast Simulation job runs in one go both Simulation and Reconstruction
- Second, the syntax of `cmsDriver.py` command is somewhat different.

If we consider a particular generator fragment, the configuration is:

```
cmsDriver.py
Configuration/GenProduction/python/ThirteenTeV/RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_py
--fileout file:RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_FASTSIM.root --mc
--eventcontent AODSIM --datatier GEN-SIM --conditions auto:mc --beamspot
Realistic8TeVCollision --step GEN,FASTSIM,HLT:GRun --python_filename
RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_pythia8_FASTSIM_cfg.py --no_exec
--customise Configuration/DataProcessing/Utils.addMonitoring -n 64
```

You can see the change in the "-s" and "datatier" options. For the rest all the above applies.

```
For Run2-CMSSW_8_0_X
```

```
cmsDriver.py
Configuration/GenProduction/python/ThirteenTeV/RSGravitonToZZ_kMpl01_M_1000_TuneCUETP8M1_13TeV_py
--conditions auto:run2_mc --fast -n 10 --era Run2_2016 --eventcontent AODSIM --relval
100000,1000 -s
GEN,SIM,RECOBEFMIX,DIGI:pdigi_valid,L1,DIGI2RAW,L1Reco,RECO,EI,HLT:@relval2016, --datatier
AODSIM --beamspot Realistic50ns13TeVCollision

For Run2-CMSSW_10_2_X

cmsDriver.py Configuration/GenProduction/python/BPH-RunIIFall18GS-00170-fragment.py --fileout fil

cmsRun  BsToPhiMuMuRunIIAutumn18FSPremix_cfg.py
```

Walk-through Example Full Simulation Configuration File

If you wish to learn details about **CMSSW configuration language** in general, please visit the corresponding section in this WorkBook.

In this document, we will provide a partial walk-through the example configuration, concentrating on details that are specific to event generation and detector simulation.

First of all, please notice this line:

```
process.source = cms.Source("EmptySource")
```

You will need to use EmptySource if you wish to generate events "from scratch", using one of the multi-purpose event generators. There are several types of "Sources" in CMSSW, that are useful for other types of event generation, or for processing pre-generated events through further steps.

Next block in the configuration file that will be of interest to you is the one that starts with

```
process.generator = cms.EDFilter("Pythia8GeneratorFilter",
```

followed by a long string of configuration commands. This is the software module of CMSSW that interfaces Pythia8 multi-purpose event generator. It contains:

- blocks of commands which are the default for all MC samples (pythia8CommonSettings and pythia8CUEP8M1Settings): the former defines common parameters for running Pythia8, the latter defines the underlying event tune.
- a block of command (process parameters) which are specific to this process. In the case of the graviton, the five strings in order: 1) enables the production of G, 2) defines the kappa parameter in RS theory, 3) sets the G mass to 1 TeV and 4) 5) switches off all G decays **except** those containing a Z (i.e. ZZ only).

Please note that in the event processing chain this module is labeled as "generator". Please note that "generator" is a single label to apply to the event generation step, no matter what event generator you choose to employ. It is by this particular label that subsequent steps in the event processing chain will find generator-level particles for further processing.

Towards the beginning of the configuration file you will see other topics of interest to you:

```
process.load('Configuration.StandardSequences.SimIdeal_cff')
process.load('Configuration.StandardSequences.Digi_cff')
```

The first one is a "master" configuration fragment for Geant4-based detector simulation, and the second one

brings together several software components to model electronics response in all parts of the CMS detector. The step that simulates passage of particles through CMS detector is labeled "psim". The sequence of steps to perform digitization is collectively labeled "pdigi".

Towards the end of the configuration file you will notice how these labels are used to include these software in the processing chain:

```
process.simulation_step = cms.Path(process.psim)
process.digitisation_step = cms.Path(process.pdigi)
```

and ordered for execution:

```
process.schedule = cms.Schedule(...,process.simulation_step,process.digitisation_step,...)
```

Please be advised that there are other software components needed to properly run CMS detector simulation and digitization, that are also included in the example configuration file. We will cover more details in the WorkBookSimDigi and its daughter materials.

Please note that these are pre-fabricated "building blocks" that are available as part of the Configuration/StandardSequences package of CMSSW. In its turn each configuration fragment uses other **standard**, pre-fabricated configuration includes and fragments, in order to configure CMSSW simulation and digitization software with the **CMS-approved setting**. If you wish to learn more details about the underlying software components or how to reconfigure one or another module to your specific needs, please visit WorkBookSimDigi and its daughter materials.

Walk-through Example Fast Simulation Configuration File

Generator-wise the content of the Fast Simulation configuration file is very much similar to what is described in a previous chapter for Full Simulation configuration.

In this case the only module performing Simulation and Digitization together is:

```
process.load('FastSimulation.Configuration.FamosSequences_cff')
```

There are also several explicit replacements of the default parameters, which set number of in-time pileup events, misalignment options, switch on/off particular subdetector simulation etc.

```
process.famosPileup.PileUpSimulator.averageNumber = 0
process.famosSimHits.SimulateCalorimetry = True
...
```

And at the end of the configuration file, a sequence of modules execution is defined with the job schedule commands (`process.schedule...`), so that generation and simulation parts are executed first, then follows a trigger part, and finally reconstruction and output ones.

Review status

Reviewer/Editor and Date (copy from screen)	Comments
ElizaMelo - 2019-09-30	Instructions for Run2 (CMSSW102X) added.
ElizaMelo - 2017-08-20	Instructions for Run2 added and updated for latest releases .
RobertoCovarelli - 2015-03-03	Major rewrite for Run2.

AnneHeavey - 15 Sep 2006	Separated this content out from the generation page, per Julia's request.

Responsible: JuliaYarba
Last reviewed by: DanielElvira - 15 Nov 2006

This topic: CMSPublic > WorkBookGenIntro
Topic revision: r57 - 2020-01-16 - ElizaMelo