

Using Sherpa in CMSSW

Philipp Millet, Sebastian Thüer

Sherpa Tutorial
29.08.2016



Bundesministerium
für Bildung
und Forschung



RWTHAACHEN
UNIVERSITY

INTRODUCTION

- Sherpa Tutorial Twiki
- content
 - presentation part
 - Sherpa in CMSSW
 - SherpalInterface
 - Sherpa workflow for event production
 - hands-on part
 - exercise 1 – producing events
 - exercise 2 – event weights, scale- and PDF-variations
- acknowledgments: big thanks to

Vitaliano Ciulli, Qiang Li, Marek Schoenherr and Sebastian Thüer
for their help in preparing / testing the exercises

SHERPA IN CMSSW – OVERVIEW

- Sherpa is included in CMSSW as external package
 - <https://github.com/cms-externals/sherpa>
 - version coupled to CMSSW release
 - newest versions often only present in newest releases
 - newest Sherpa version: 2.2.1
 - current versions in CMSSW:
 - CMSSW 71X – Sherpa v2.2.0
 - v2.2.0 for releases \geq CMSSW_7_1_23
 - CMSSW 80X – Sherpa v2.2.0
 - CMSSW 81X – Sherpa v2.2.1
 - v2.2.1 for (pre)releases \geq CMSSW_8_1_0_pre10
- uses a couple of additional external packages:
 - HepMC, LHAPDF, Sqlite, FastJet, ... (used by many others)
 - BlackHat, Openloops, ... (used only by Sherpa)

SHERPA IN CMSSW – SPECFILE

- Sherpa can be installed with different options / externals
- to find the settings for a given CMSSW release:
 - go to <https://github.com/cms-sw/cmsdist/>
 - select corresponding branch and look at `sherpa.spec`
 - e.g. for `CMSSW_8_1_X`

```
### RPM external sherpa 2.2.1 ← Sherpa version
[...]
%build
./configure --prefix=%i --enable-analysis --disable-silent-rules \
    --enable-fastjet=$FASTJET_ROOT --enable-mcfm=$MCFM_ROOT \
    --enable-hepmc2=$HEPMC_ROOT --enable-lhapdf=$LHAPDF_ROOT \
    --enable-blackhat=$BLACKHAT_ROOT \ --enable-pyext --enable-ufo \
    ${OPENLOOPSP_ROOT}+--enable-openloops=$OPENLOOPSP_ROOT} \
    --with-sqlite3=$SQLITE_ROOT \
    CXX="%cms_cxx" \
    CXXFLAGS="-fuse-cxa-atexit $ARCH_CMSPALF %cms_cxxflags -I$LHAPDF_ROOT/include - \
    I$BLACKHAT_ROOT/include -I$OPENSSL_ROOT/include" \
    LDFLAGS="-ldl -L$BLACKHAT_ROOT/lib/blackhat -L$QD_ROOT/lib -L$OPENSSL_ROOT/lib"
[...]
```

← configured with LHAPDF, BlackHat, OpenLoops, UFO Interface, ...

THE CMSSW SHERPA INTERFACE – OVERVIEW

- the Sherpa Interface is part of CMSSW

- source code:

https://github.com/cms-sw/cmssw/tree/CMSSW_8_1_X/GeneratorInterface/SherpaInterface

- purpose:

- generate events with Sherpa
 - turn events into compliant format
 - feed output into CMSSW

- Sherpa-CMSSW workflow

- phase-space integration and library creation
 - event generation
 - performed by interface
 - can be parallelized on grid

← details next slides

← details later

THE CMSSW SHERPA INTERFACE – INTERFACE SCRIPTS

- phase-space integration / creation of libraries can be very time consuming
 - depends on complexity of process, additional jets, ...
 - can take hours / days / weeks
 - must not repeat this step for every job in central production
- idea
 - do this once locally and reuse for every job
 - store results in tarball
- two scripts are provided for this purpose
 - `MakeSherpaLibs.sh`
 - perform phase-space integration
 - create libraries
 - `PrepareSherpaLibs.sh`
 - collect all needed results in `sherpack`
 - create python config for interface

```
SherpaInterface/
|-- BuildFile.xml
|-- data
|   |-- MakeSherpaLibs.sh
|   |-- PrepareSherpaLibs.sh
|   `-- [...]
|-- doc
|   '-- [...]
|-- interface
|   '-- [...]
|-- python
|   '-- [...]
`-- src
    '-- [...]
```

THE CMSSW SHERPA INTERFACE – INTEGRATION AND LIBRARIES

- phase-space integration and creation of libraries is done with `MakeSherpaLibs.sh`
 - depending on your settings, several steps are needed before producing events
 - all steps executed automatically by `MakeSherpaLibs.sh` depending on runcard
 - input: Sherpa runcard
 - output: several tarballs containing libraries, cross-sections, logs, cards, etc..
 - usage: `./MakeSherpaLibs.sh -h`

MakeSherpaLibs version 4.4

options:

```
[..]  
-p process  SHERPA process/dataset name ( XXX )           ← name tag  
-o option   library/cross section options ( LIBS )  
             [ 'LBCR' : generate libraries and cross sections ]  
             [ 'LIBS' : generate libraries only ]  
             [ 'CRSS' : generate cross sections, needs libraries! ]  
             [ 'EVTS' : generate events, needs libs + crss. sec.! ]  
-v          verbose mode ( FALSE )  
[..]
```

- convention for runcard name:
`Run.dat_XXX`

THE CMSSW SHERPA INTERFACE – INTEGRATION AND LIBRARIES

- phase-space integration and creation of libraries is done with `MakeSherpaLibs.sh`
 - depending on your settings, several steps are needed before producing events
 - all steps executed automatically by `MakeSherpaLibs.sh` depending on runcard
 - input: Sherpa runcard
 - output: several tarballs containing libraries, cross-sections, logs, cards, etc..
 - usage: `./MakeSherpaLibs.sh -h`

MakeSherpaLibs version 4.4

options:

```
[..]  
-p process  SHERPA process/dataset name ( XXX )  
-o option   library/cross section options ( LIBS )  
             [ 'LBCR' : generate libraries and cross sections ]  
             [ 'LIBS' : generate libraries only ]  
             [ 'CRSS' : generate cross sections, needs libraries! ]  
             [ 'EVTS' : generate events, needs libs + crss. sec.! ]  
-v          verbose mode ( FALSE )  
[..]
```



steps to be executed:

- when starting from runcard:
`LBCR`
- when results partly present
consider other options

THE CMSSW SHERPA INTERFACE – INTEGRATION AND LIBRARIES

- phase-space integration and creation of libraries is done with `MakeSherpaLibs.sh`
 - depending on your settings, several steps are needed before producing events
 - all steps executed automatically by `MakeSherpaLibs.sh` depending on runcard
 - input: Sherpa runcard
 - output: several tarballs containing libraries, cross-sections, logs, cards, etc..
 - usage: `./MakeSherpaLibs.sh -h`

MakeSherpaLibs version 4.4

options:

```
[..]  
-p process  SHERPA process/dataset name ( XXX )  
-o option   library/cross section options ( LIBS )  
             [ 'LBCR' : generate libraries and cross sections ]  
             [ 'LIBS' : generate libraries only           ]  
             [ 'CRSS' : generate cross sections, needs libraries! ]  
             [ 'EVTS' : generate events, needs libs + crss. sec.! ]  
-v          verbose mode ( FALSE )  
[..]
```

← print additional information
recommended to see progress

THE CMSSW SHERPA INTERFACE – CREATING SHERPACKS

- SherpalInterface needs two inputs:
 - sherpack containing integration results and libraries from Sherpa
 - python configuration fragment
- both inputs can be created with `PrepareSherpaLibs.sh`
 - collects output from `MakeSherpaLibs.sh` into sherpack
 - creates python configuration based on runcard and sherpack
 - usage: `./PrepareSherpaLibs.sh -h`

`PrepareSherpaLibs` version 4.2

options: [..]

```
-p process  SHERPA dataset/process name ( XXX )           ← name tag (as before)
              [...]
-e filename (optional) name of extended weight list file ( )
              (example file in GeneratorInterface/SherpalInterface/python/ExtendedSherpaWeights_cfi.py)
-h          display this help and exit
```

THE CMSSW SHERPA INTERFACE – CREATING SHERPACKS

- SherpalInterface needs two inputs:
 - sherpack containing integration results and libraries from Sherpa
 - python configuration fragment
- both inputs can be created with `PrepareSherpaLibs.sh`
 - collects output from `MakeSherpaLibs.sh` into sherpack
 - creates python configuration based on runcard and sherpack
 - usage: `./PrepareSherpaLibs.sh -h`

`PrepareSherpaLibs` version 4.2

options: [..]

```
-p process  SHERPA dataset/process name ( XXX )
            [..]
-e filename (optional) name of extended weight list file ( )
            (example file in GeneratorInterface/SherpalInterface/python/ExtendedSherpaWeights_cfi.py)
-h          display this help and exit
```



weights file for scale/PDF variations
(details later)

THE CMSSW SHERPA INTERFACE – WORKFLOW

- Sherpa produces fully decayed, merged and showered events
 - no intermediate LHE step
 - no need of Pythia for decays or parton shower
- main ingredients of the interface
 - SherpaHadronizer which is a [BaseHadronizer](#)
 - SherpaGeneratorFilter which is a [edm::GeneratorFilter](#)
- workflow
 - within hadronizer:
 - generate event with Sherpa
 - convert event to HepMC (using Sherpa method)
 - pass event over to GeneratorFilter
 - GeneratorFilter turns HepMC into EDM event

```
SherpaInterface/
|-- BuildFile.xml
|-- data
|   '-- [...]
|-- doc
|   '-- [...]
|-- interface
|   '-- SherpackFetcher.h
|       '-- SherpackUtilities.h
|-- python
|   '-- [...]
`-- src
    |-- SherpaHadronizer.cc
    |-- SherpackFetcher.cc
    '-- SherpackUtilities.cc
```

THE CMSSW SHERPA INTERFACE – PYTHON FRAGMENT

- Sherpa Interface steered by python fragment
- automatically generated by scripts provided with interface
- example file:

```
generator = cms.EDFilter("SherpaGeneratorFilter",
[...]
SherpaProcess = cms.string('7TeV_WtoMNu_1jLO'), ← name tag for process
SherpackLocation = cms.string('.'),
SherpackChecksum =
cms.string('b0cc92af17b68561cdac88690fb0c8bd'),
FetchSherpack = cms.bool(False),
[...]
SherpaParameters = cms.PSet(
    parameterSets = cms.vstring(
        "MPI_Cross_Sections",
        "Run"
    ),
    MPI_Cross_Sections = cms.vstring("[.."]),
    Run = cms.vstring("[COPY OF THE RUNCARD]"),
[...]
```

THE CMSSW SHERPA INTERFACE – PYTHON FRAGMENT

- Sherpa Interface steered by python fragment
- automatically generated by scripts provided with interface
- example file:

```
generator = cms.EDFilter("SherpaGeneratorFilter",
[...]
SherpaProcess = cms.string('7TeV_WtoMNu_1jLO'),
SherpackLocation = cms.string('.'),
SherpackChecksum =
cms.string('b0cc92af17b68561cdac88690fb0c8bd'),
FetchSherpack = cms.bool(False),
[...]
SherpaParameters = cms.PSet(
    parameterSets = cms.vstring(
        "MPI_Cross_Sections",
        "Run"
    ),
    MPI_Cross_Sections = cms.vstring("[.."]),
    Run = cms.vstring("[COPY OF THE RUNCARD]"),
[...]
```



location of tarball

- '.' for standalone production
- '/cvmfs/..' for central production

THE CMSSW SHERPA INTERFACE – PYTHON FRAGMENT

- Sherpa Interface steered by python fragment
- automatically generated by scripts provided with interface
- example file:

```
generator = cms.EDFilter("SherpaGeneratorFilter",
[...]
SherpaProcess = cms.string('7TeV_WtoMNu_1jLO'),
SherpackLocation = cms.string('.'),
SherpackChecksum =
cms.string('b0cc92af17b68561cdac88690fb0c8bd'),
FetchSherpack = cms.bool(False),
[...]
SherpaParameters = cms.PSet(
    parameterSets = cms.vstring(
        "MPI_Cross_Sections",
        "Run"
    ),
    MPI_Cross_Sections = cms.vstring("[.."]),
    Run = cms.vstring("[COPY OF THE RUNCARD]"),
[...]
```

- ← copy sherpack to current directory
- false for standalone production
 - true central production

THE CMSSW SHERPA INTERFACE – PYTHON FRAGMENT

- Sherpa Interface steered by python fragment
- automatically generated by scripts provided with interface
- example file:

```
generator = cms.EDFilter("SherpaGeneratorFilter",
[...]
SherpaProcess = cms.string('7TeV_WtoMNu_1jLO'),
SherpackLocation = cms.string('.'),
SherpackChecksum =
cms.string('b0cc92af17b68561cdac88690fb0c8bd'),           ← checksum of tarball
FetchSherpack = cms.bool(False),
[...]
SherpaParameters = cms.PSet(
    parameterSets = cms.vstring(
        "MPI_Cross_Sections",
        "Run"
    ),
    MPI_Cross_Sections = cms.vstring("[.."]),
    Run = cms.vstring("[COPY OF THE RUNCARD]"),
[...]
```

THE CMSSW SHERPA INTERFACE – PYTHON FRAGMENT

- Sherpa Interface steered by python fragment
- automatically generated by scripts provided with interface
- example file:

```
generator = cms.EDFilter("SherpaGeneratorFilter",
[...]
SherpaProcess = cms.string('7TeV_WtoMNu_1jLO'),
SherpackLocation = cms.string('.'),
SherpackChecksum =
cms.string('b0cc92af17b68561cdac88690fb0c8bd'),
FetchSherpack = cms.bool(False),
[...]
SherpaParameters = cms.PSet(
    parameterSets = cms.vstring(
        "MPI_Cross_Sections",
        "Run"
    ),
    MPI_Cross_Sections = cms.vstring("[.."]),
    Run = cms.vstring("[COPY OF THE RUNCARD]") ➔ runcard transferred to SherpalInterface
[...]
```

STANDALONE SHERPA

- sometimes it may be useful to run Sherpa standalone
- Sherpa executable is called **Sherpa**
- to find location of Sherpa executable run: **scram tool info Sherpa**
- output:

Tool info as configured in location /afs/cern.ch/work/m/millet/public/CMSSW_8_1_0_pre10
+++++

Name : sherpa

Version : 2.2.1 ← version

++++++

SCRAM_PROJECT=no

SHERPA_BASE=/cvmfs/cms.cern.ch/slc6_amd64_gcc530/external/sherpa/2.2.1

BINDIR=/cvmfs/cms.cern.ch/slc6_amd64_gcc530/external/sherpa/2.2.1/bin ← location executable

[..]

Exercise I

Producing Events

EXERCISE I – OVERVIEW

- goals of this exercise:
 - perform full CMSSW Sherpa workflow
 - write runcard
 - phase-space integration / library creation
 - create sherpack
 - adapt configuration fragment
 - produce events
 - analyze events with Rivet and compare with CMS results
- example process: $W \rightarrow \mu \nu$
- instructions can be found at [Sherpa Tutorial Twiki](#)
- prerequisites:
 - laptop / computer
 - lxplus (or similar cluster) access

Exercise II

Event Weights, Scale- and PDF-Variations

EXERCISE II – OVERVIEW

- goals of this exercise:
 - modify runcard to include scale / PDF variations
 - perform CMSSW Sherpa workflow starting from integrated results
 - create sherpack including scale / PDF variations
 - adapt configuration fragment
 - produce events
 - write EDAnalyzer
 - loop over genparticles
 - access weights and weight names from variations
- instructions can be found at [Sherpa Tutorial Twiki](#)
- prerequisites:
 - laptop / computer
 - Ixplus (or similar cluster) access

SCALE- AND PDF-VARIATIONS

- Sherpa is able to calculate scale / PDF-variations on-the-fly
- each variation will lead to an additional weight stored with the event
- variations have to be specified in the run card
- the Sherpa Interface has the possibility to
 - store weight names
 - reorder weights
- for this purpose a python configuration file has to be given to `PrepareSherpaLibs.sh`
 - will be included in the `sherpack`
 - tells interface which names to store / how to order weights
- example

```
SherpaWeightsBlock = cms.PSet(  
    SherpaWeights = cms.vstring( 'Weight',  
        [..]  
    ),  
    SherpaVariationWeights = cms.vstring( 'MUR0.5_MUF0.5_PDF261000',  
        [..]  
    ),  
)
```



WRITING A SHERPA ANALYZER – BASICS

- goal: create basic analyzer to loop over produced events
 - access genparticle and genjets
 - access names and values of variation weights
- first steps:
 - go to `src/` directory of your CMSSW release
 - create subdirectory (e.g. `mkdir DemoAnalyzer`)
 - create EDAnalyzer skeleton with `mkedanlzr DemoAnalyzer`
 - output:

```
DemoAnalyzer/
|   plugins/
|   |-- BuildFile.xml
|   |-- DemoAnalyzer.cc
|   python/
|   |-- CfiFile_cfi.py
|   |-- ConfFile_cfg.py
|   test/
|   doc/
```

← actual analyzer

← python configuration file

WRITING A SHERPA ANALYZER – GENPARTICLES

- goal: access GenParticles in `DemoAnalyzer::analyze(..)` function
 - add `#include "DataFormats/HepMCCandidate/interface/GenParticle.h` to includes
 - register access to GenParticles in constructor:
 - `consumes<std::vector<reco::GenParticle>>(edm::InputTag("genParticles"));`
 - add `<use name="SimDataFormats/GeneratorProducts"/>` to `BuildFile.xml`
 - access GenParticles in `DemoAnalyzer::analyze(..)`

```
Handle< std::vector<reco::GenParticle> > genParticles;
iEvent.getByLabel("genParticles", genParticles);
for(size_t i = 0; i < genParticles->size(); ++ i) {
    const reco::GenParticle & p = (*genParticles)[i];
    int id = p.pdgId();
    if (abs(id) == 13 && p.isPromptFinalState()) {
        // DO STUFF
    }
}
```

- similar steps for GenJets

WRITING A SHERPA ANALYZER – WEIGHTS

- goal: access event weights in `DemoAnalyzer::analyze(..)` function
 - add to includes:
 - `#include "SimDataFormats/GeneratorProducts/interface/GenEventInfoProduct.h"`
 - register access to `GenEventInfoProduct` in constructor:
 - `consumes<GenEventInfoProduct>(edm::InputTag("generator"));`
 - access event weights in `DemoAnalyzer::analyze(..)`

```
Handle<GenEventInfoProduct> genInfo;  
Event.getByLabel("generator", genInfo);  
std::cout << "There are " << genInfo->weights().size() << " weights for this event." << std::endl;  
double weight=genInfo->weights()[0];
```

WRITING A SHERPA ANALYZER – WEIGHT NAMES

- goal: access event names in `DemoAnalyzer`
 - weight names are not stored for every event but once per LumiSection in `GenLumiInfoHeader`
 - register access to `GenEventInfoProduct` in constructor:
 - `consumes<GenLumiInfoHeader,edm::InLumi>(edm::InputTag("generator"));`
 - need `EDAnalyzer` to be aware of LumiSections
 - change class definition

[documentation](#)

```
class DemoAnalyzer : public edm::one::EDAnalyzer<edm::one::SharedResources> { ... }  
↓  
class DemoAnalyzer : public edm::one::EDAnalyzer<edm::one::SharedResources,edm::one::WatchLuminosityBlocks> { ... }  


- implement beginLuminosityBlock() and endLuminosityBlock() functions



```
void DemoAnalyzer::beginLuminosityBlock(const edm::LuminosityBlock& iRun, const edm::EventSetup& iSetup){
 using namespace edm;
 Handle<GenLumiInfoHeader> genLumiHeader;
 iRun.getByLabel("generator", genLumiHeader);
 LogInfo("DemoAnalyzer") << "There are " << genLumiHeader->weightNames().size() << " total weights per event:";
 for (unsigned int i=0; i<genLumiHeader->weightNames().size(); i++) {
 LogInfo("DemoAnalyzer") << "weight " << i << " " << genLumiHeader->weightNames()[i];
 }
}
```


```

THANK YOU FOR YOUR ATTENTION

Backup