# Sherpa Manual

*Release 3.0.0*

**Sherpa Team**

**Feb 10, 2021**

# CONTENTS

Welcome to the Sherpa manual. The manual for versions prior to `3.0.0` can be found here.

CHAPTER

ONE

# INTRODUCTION

Sherpa is a Monte Carlo event generator for the Simulation of High-Energy Reactions of Particles in lepton-lepton, lepton-photon, photon-photon, lepton-hadron and hadron-hadron collisions. This document provides information to help users understand and apply Sherpa for their physics studies. The event generator is introduced, in broad terms, and the installation and running of the program are outlined. The various options and parameters specifying the program are compiled, and their meanings are explained. This document does not aim at giving a complete description of the physics content of Sherpa. To this end, the authors refer the reader to the original publications, [GHK+09] and [B+19].



## 1.1 Introduction to Sherpa

Sherpa [B+19] is a Monte Carlo event generator that provides complete hadronic final states in simulations of high-energy particle collisions. The produced events may be passed into detector simulations used by the various experiments. The entire code has been written in C++, like its competitors Herwig++ [B+08] and Pythia 8 [SMS08].

Sherpa simulations can be achieved for the following types of collisions:

- for lepton–lepton collisions, as explored by the CERN LEP experiments,
- for lepton–photon collisions,
- for photon–photon collisions with both photons either resolved or unresolved,
- for deep-inelastic lepton-hadron scattering, as investigated by the HERA experiments at DESY, and,
- in particular, for hadronic interactions as studied at the Fermilab Tevatron or the CERN LHC.

The list of physics processes that can be simulated with Sherpa covers all reactions in the Standard Model. Other models can be implemented either using Sherpa's own model syntax, or by using the generic interface [HKSS15] to the UFO output [DDF+12] of FeynRules [CD09],:cite:*Christensen2009jx*. The Sherpa program owes this versatility to the two inbuilt matrix-element generators, AMEGIC++ and Comix, and to it's phase-space generator Phasic [KKS02], which automatically calculate and integrate tree-level amplitudes for the implemented models. This feature enables Sherpa to be used as a cross-section integrator and parton-level event generator as well. This aspect has been extensively tested, see e.g. [GKP+04], [H+06].

As a second key feature of Sherpa the program provides an implementation of the merging approaches of [HKSS09] and [GHK+13], [HKSS13]. These algorithms yield improved descriptions of multijet production processes, which copiously appear at lepton-hadron colliders like HERA [CGH10], or hadron-hadron colliders like the Tevatron and the LHC, [KSSS04], [KSSS05], [GKS+05], [HSS10]. An older approach, implemented in previous versions of Sherpa

and known as the CKKW technique [CKKW01], [Kra02], has been compared in great detail in [A+08] with other approaches, such as the MLM merging prescription [MMP02] as implemented in Alpgen [MMP+03], Madevent [SL94], [MS03], or Helac [KP00], [PW07] and the CKKW-L prescription [Lon02], [LL05] of Ariadne [Lon92].

This manual contains all information necessary to get started with Sherpa as quickly as possible. It lists options and switches of interest for steering the simulation of various physics aspects of the collision. It does not describe the physics simulated by Sherpa or the underlying structure of the program. Many external codes can be linked with Sherpa. This manual explains how to do this, but it does not contain a description of the external programs. You are encouraged to read their corresponding documentations, which are referenced in the text. If you use external programs with Sherpa, you are encouraged to cite them accordingly.

The MCnet Guidelines apply to Sherpa. You are kindly asked to cite [B+19] if you have used the program in your work.

The Sherpa authors strongly recommend the study of the manuals and many excellent publications on different aspects of event generation and physics at collider experiments written by other event generator authors.

This manual is organized as follows: in *Basic structure* the modular structure intrinsic to Sherpa is introduced. *Getting started* contains information about and instructions for the installation of the package. There is also a description of the steps that are needed to run Sherpa and generate events. The *Input structure* is then discussed, and the ways in which Sherpa can be steered are explained. All parameters and options are discussed in *Parameters*. Advanced *Tips and tricks* are detailed, and some options for *Customization* are outlined for those more familiar with Sherpa. There is also a short description of the different *Examples* provided with Sherpa.

The construction of Monte Carlo programs requires several assumptions, approximations and simplifications of complicated physics aspects. The results of event generators should therefore always be verified and cross-checked with results obtained by other programs, and they should be interpreted with care and common sense.

## 1.2 Basic structure

Sherpa is a modular program. This reflects the paradigm of Monte Carlo event generation, with the full simulation is split into well defined event phases, based on QCD factorization theorems. Accordingly, each module encapsulates a different aspect of event generation for high-energy particle reactions. It resides within its own namespace and is located in its own subdirectory of the same name. The main module called `SHERPA` steers the interplay of all modules – or phases – and the actual generation of the events. Altogether, the following modules are currently distributed with the Sherpa framework:

**ATOOLS** This is the toolbox for all other modules. Since the Sherpa framework does not rely on CLHEP etc., the ATOOLS contain classes with mathematical tools like vectors and matrices, organization tools such as read-in or write-out devices, and physics tools like particle data or classes for the event record.

**METOOLS** In this module some general methods for the evaluation of helicity amplitudes have been accumulated. They are used in AMEGIC++, the EXTRA_XS module, and the new matrix-element generator Comix. This module also contains helicity amplitudes for some generic matrix elements, that are, e.g., used by HADRONS++. Further, METOOLS also contains a simple library of tensor integrals which are used in the PHOTONS++ matrix element corrections.

**BEAM** This module manages the treatment of the initial beam spectra for different colliders. The three options which are currently available include a monochromatic beam, which requires no extra treatment, photon emission in the Equivalent Photon Approximation (EPA) and - for the case of an electron collider - laser backscattering off the electrons, leading to photonic initial states.

**PDF** The PDF module provides access to various parton density functions (PDFs) for the proton and the photon. In addition, it hosts an interface to the LHAPDF package, which makes a full wealth of PDFs available. An (analytical) electron structure function is supplied in the PDF module as well.

**MODEL** This module sets up the physics model for the simulation. It initializes particle properties, basic physics parameters (coupling constants, mixing angles, etc.) and the set of available interaction vertices (Feynman rules). By now, there exist explicit implementations of the Standard Model (SM), its Minimal Supersymmetric extension (MSSM), the ADD model of large extra dimensions, and a comprehensive set of operators parametrizing anomalous triple and quartic electroweak gauge boson couplings. An Interface to FeynRules is also available.

**EXTRA_XS** In this module a (limited) collection of analytic expressions for simple 2->2 processes within the SM are provided together with classes embedding them into the Sherpa framework. This also includes methods used for the definition of the starting conditions for parton-shower evolution, such as colour connections and the hard scale of the process.

**AMEGIC++** AMEGIC++ [KKS02] is Sherpa's original matrix-element generator. It employs the method of helicity amplitudes [KS85], [BMM94] and works as a generator, which generates generators: During the initialization run the matrix elements for a given set of processes, as well as their specific phase-space mappings are created by AMEGIC++. Corresponding C++ sourcecode is written to disk and compiled by the user using the `makelibs` script or scons. The produced libraries are linked to the main program automatically in the next run and used to calculate cross sections and to generate weighted or unweighted events. AMEGIC++ has been tested for multi-particle production in the Standard Model [GKP+04]. Its MSSM implementation has been validated in [H+06].

**COMIX** Comix is a multi-leg tree-level matrix element generator, based on the color dressed Berends-Giele recursive relations [DHM06]. It employs a new algorithm to recursively compute phase-space weights. The module is a useful supplement to older matrix element generators like AMEGIC++ in the high multiplicity regime. Due to the usage of colour sampling it is particularly suited for an interface with parton shower simulations and can hence be easily employed for the ME-PS merging within Sherpa. It is Sherpa's default large multiplicity matrix element generator for the Standard Model.

**PHASIC++** All base classes dealing with the Monte Carlo phase-space integration are located in this module. For the evaluation of the initial-state (laser backscattering, initial-state radiation) and final-state integrals, the adaptive multi-channel method of [KP94], [BPK94] is used by default together with a Vegas optimization [Lep] of the single channels. In addition, final-state integration accomplished by Rambo [KSE86], Sarge [DvHK00] and HAAG [vHP02] is supported.

**CSSHOWER++** This is the module hosting Sherpa's default parton shower, which was published in [SK08b]. The corresponding shower model was originally proposed in [NS05], [NS]. It relies on the factorisation of real-emission matrix elements in the CS subtraction framework [CS97], [CDST02]. There exist four general types of CS dipole terms that capture the complete infrared singularity structure of next-to-leading order QCD amplitudes. In the large-N_C limit, the corresponding splitter and spectator partons are always adjacent in colour space. The dipole functions for the various cases, taken in four dimensions and averaged over spins, are used as shower splitting kernels.

**DIRE** This is the module hosting Sherpa's alternative parton shower [HP]. In the Dire model, the ordering variable exhibits a symmetry in emitter and spectator momenta, such that the dipole-like picture of the evolution can be re-interpreted as a dipole picture in the soft limit. At the same time, the splitting functions are regularized in the soft anti-collinear region using partial fractioning of the soft eikonal in the Catani-Seymour approach [CS97], [CDST02]. They are then modified to satisfy the sum rules in the collinear limit. This leads to an invariant formulation of the parton-shower algorithm, which is in complete analogy to the standard DGLAP case, but generates the correct soft anomalous dimension at one-loop order.

**AMISIC++** AMISIC++ contains classes for the simulation of multiple parton interactions according to [SvZ87]. In Sherpa the treatment of multiple interactions has been extended by allowing for the simultaneous evolution of an independent parton shower in each of the subsequent (semi-)hard collisions. The beam–beam remnants are organized such that partons which are adjacent in colour space are also adjacent in momentum space. The corresponding classes for beam remnant handling reside in the PDF and SHERPA modules.

**AHADIC++** AHADIC++ is Sherpa's hadronization package, for translating the partons (quarks and gluons) into primordial hadrons, to be further decayed in HADRONS++. The algorithm bases on the cluster fragmentation ideas

presented in [Got83], [Got84], [Web84], [GM87] and implemented in the Herwig family of event generators. The actual Sherpa implementation, based on [WKS04], differs from the original model in several respects.

**HADRONS++** HADRONS++ is the module for simulating hadron and tau-lepton decays. The resulting decay products respect full spin correlations (if desired). Several matrix elements and form-factor models have been implemented, such as the Kühn-Santamaría model, form-factor parametrizations from Resonance Chiral Theory for the tau and form factors from heavy quark effective theory or light cone sum rules for hadron decays.

**PHOTONS++** The PHOTONS++ module holds routines to add QED radiation to hadron and tau-lepton decays. This has been achieved by an implementation of the YFS algorithm [YFS61]. The structure of PHOTONS++ is such that the formalism can be extended to scattering processes and to a systematic improvement to higher orders of perturbation theory [SK08a]. The application of PHOTONS++ therefore accounts for corrections that usually are added by the application of PHOTOS [Ba94] to the final state.

**SHERPA** Finally, SHERPA is the steering module that initializes, controls and evaluates the different phases during the entire process of event generation. All routines for the combination of truncated showers and matrix elements, which are independent of the specific matrix element and parton shower generator are found in this module.

The actual executable of the Sherpa generator can be found in the subdirectory `<prefix>/bin/` and is called `Sherpa`. To run the program, input files have to be provided in the current working directory or elsewhere by specifying the corresponding path, see *Input structure*. All output files are then written to this directory as well.

CHAPTER

TWO

# GETTING STARTED



## 2.1 Installation

Sherpa is distributed as a tarred and gzipped file named `SHERPA-MC-<VERSION>.tar.gz`, and can be unpacked in the current working directory with

```
$ tar -zxf SHERPA-MC-<VERSION>.tar.gz
```

Alternatively, it can also be accessed via Git through the location specified on the download page. In that case, before continuing, it is necessary to construct the build scripts by running `autoreconf -i` once after cloning the Git repo.

To guarantee successful installation, the following tools should be available on the system:

- C++ and Fortran compilers (e.g. from the gcc suite)
- make
- SQLite 3 (including the -dev package if installed through a package manager)

If SQLite is installed in a non-standard location, please specify the installation path using option `--with-sqlite3=/path/to/sqlite`. If SQLite is not installed on your system, the Sherpa configure script provides the fallback option of installing it into the same directory as Sherpa itself. To do so, please run configure with

option `--with-sqlite3=install` (This may not work if you are cross-compiling using `--host`. In this case, please install SQLite by yourself and reconfigure using `--with-sqlite3=/path/to/sqlite`).

Compilation and installation proceed through the following commands if you use the distribution tarball:

```
$ ./configure [options]
$ make install
```

To install from a cloned git repository run:

```
$ autoreconf -i
$ mkdir build && cd build
$ ../configure [options]
$ make install
```

If not specified differently, the directory structure after installation is organized as follows

**`$(prefix)/bin`** Sherpa executable and scripts

**`$(prefix)/include`** headers for process library compilation

**`$(prefix)/lib`** basic libraries

**`$(prefix)/share`** PDFs, Decaydata, fallback run cards

The installation directory `$(prefix)` can be specified by using the `./configure --prefix /path/to/installation/target` directive and defaults to the current working directory.

If Sherpa has to be moved to a different directory after the installation, one has to set the following environment variables for each run:

- `SHERPA_INCLUDE_PATH=$newprefix/include/SHERPA-MC`
- `SHERPA_SHARE_PATH=$newprefix/share/SHERPA-MC`
- `SHERPA_LIBRARY_PATH=$newprefix/lib/SHERPA-MC`
- `LD_LIBRARY_PATH=$SHERPA_LIBRARY_PATH:$LD_LIBRARY_PATH`

Sherpa can be interfaced with various external packages, e.g. HepMC, for event output, or LHAPDF, for PDFs. For this to work, the user has to pass the appropriate commands to the configure step. This is achieved as shown below:

```
$ ./configure --enable-hepmc2=/path/to/hepmc2 --enable-lhapdf=/path/to/lhapdf
```

Here, the paths have to point to the top level installation directories of the external packages, i.e. the ones containing the `lib/`, `share/`, ... subdirectories.

For a complete list of possible configuration options run `./configure --help`

The Sherpa package has successfully been compiled, installed and tested on SuSE, RedHat / Scientific Linux and Debian / Ubuntu Linux systems using the GNU C++ compiler versions 3.2, 3.3, 3.4, and 4.x as well as on Mac OS X 10 using the GNU C++ compiler version 4.0. In all cases the GNU FORTRAN compiler g77 or gfortran has been employed.

If you have multiple compilers installed on your system, you can use shell environment variables to specify which of these are to be used. A list of the available variables is printed with

```
$ ./configure --help
```

in the Sherpa top level directory and looking at the last lines. Depending on the shell you are using, you can set these variables e.g. with export (bash) or setenv (csh). Examples:

```
export CXX=g++-3.4
export CC=gcc-3.4
export CPP=cpp-3.4
```

### 2.1.1 Installation on Cray XE6 / XK7

Sherpa has been installed successfully on Cray XE6 and Cray XK7. The following configure command should be used

```
$ ./configure <your options> --enable-mpi --host=i686-pc-linux CC=CC CXX=CC FC='ftn -
↪fPIC' LDFLAGS=-dynamic
```

Sherpa can then be run with

```
$ aprun -n <nofcores> <prefix>/bin/Sherpa -lrun.log
```

The modularity of the code requires setting the environment variable `CRAY_ROOTFS`, cf. the Cray system documentation.

### 2.1.2 Installation on IBM BlueGene/Q

Sherpa has been installed successfully on an IBM BlueGene/Q system. The following configure command should be used

```
$ ./configure <your options> --enable-mpi --host=powerpc64-bgq-linux CC=mpic++␣
↪CXX=mpic++ FC='mpif90 -fPIC -funderscoring' LDFLAGS=-dynamic
```

Sherpa can then be run with

```
$ qsub -A <account> -n <nofcores> -t 60 --mode c16 <prefix>/bin/Sherpa -lrun.log
```

### 2.1.3 MacOS Installation

Since it is more complicated to set up the necessary compiler environment on a Mac we recommend using a package manager to install Sherpa and its dependencies. David Hall is hosting a repository for Homebrew packages at: http://davidchall.github.io/homebrew-hep/

In case you are compiling yourself, please be aware of the following issues which have come up on Mac installations before:

- On 10.4 and 10.5 only gfortran is supported, and you will have to install it e.g. from HPC
- If you want to reconfigure, i.e. run the command `autoreconf` or `(g)libtoolize`, you have to make sure that you have a recent version of GNU libtool (>=1.5.22 has been tested). Don't confuse this with the native non-GNU libtool which is installed in `/usr/bin/libtool` and of no use! Also make sure that your autools (autoconf >= 2.61, automake >= 1.10 have been tested) are of recent versions. All this should not be necessary though, if you only run `configure`.
- Make sure that you don't have two versions of g++ and libstdc++ installed and being used inconsistently. This appeared e.g. when the gcc suite was installed through Fink to get gfortran. This caused Sherpa to use the native MacOS compilers but link the libstdc++ from Fink (which is located in /sw/lib). You can find out which libraries are used by Sherpa by running `otool -L bin/Sherpa`

## 2.2 Running Sherpa

The `Sherpa` executable resides in the directory `<prefix>/bin/` where `<prefix>` denotes the path to the Sherpa installation directory. The way a particular simulation will be accomplished is defined by several parameters, which can all be listed in a common file, or data card (Parameters can be alternatively specified on the command line; more details are given in *Input structure*). This steering file is called `Sherpa.yaml` and some example setups (i.e. `Sherpa.yaml` files) are distributed with the current version of Sherpa. They can be found in the directory `<prefix>/share/SHERPA-MC/Examples/`, and descriptions of some of their key features can be found in the section *Examples*.

---

**Note:** It is not in general possible to reuse steering files from previous Sherpa versions. Often there are small changes in the parameter syntax of the files from one version to the next. These changes are documented in our manuals. In addition, update any custom Decaydata directories you may have used (and reapply any changes which you might have applied to the old ones), see *Hadron decays*.

---

The very first step in running Sherpa is therefore to adjust all parameters to the needs of the desired simulation. The details for doing this properly are given in *Parameters*. In this section, the focus is on the main issues for a successful operation of Sherpa. This is illustrated by discussing and referring to the parameter settings that come in the example steering file `./Examples/V_plus_Jets/LHC_ZJets/Sherpa.yaml`, cf. *Z+jets production*. This is a simple configuration created to show the basics of how to operate Sherpa. **It should be stressed that this steering file relies on many of Sherpa's default settings, and, as such, you should understand those settings before using it to look at physics.** For more information on the settings and parameters in Sherpa, see *Parameters*, and for more examples see the *Examples* section.

### 2.2.1 Process selection and initialization

Central to any Monte Carlo simulation is the choice of the hard processes that initiate the events. These hard processes are described by matrix elements. In Sherpa, the selection of processes happens in the `PROCESSES` part of the steering file. Only a few 2->2 reactions have been hard-coded. They are available in the EXTRA_XS module. The more usual way to compute matrix elements is to employ one of Sherpa's automated tree-level generators, AMEGIC++ and Comix, see *Basic structure*. If no matrix-element generator is selected, using the *ME_GENERATORS* tag, then Sherpa will use whichever generator is capable of calculating the process, checking Comix first, then AMEGIC++ and then EXTRA_XS. Therefore, for some processes, several of the options are used. In this example, however, all processes will be calculated by Comix.

To begin with the example, the Sherpa run has to be started by changing into the `<prefix>/share/SHERPA-MC/Examples/V_plus_Jets/LHC_ZJets/` directory and executing

```
$ <prefix>/bin/Sherpa
```

You may also run from an arbitrary directory, employing `<prefix>/bin/Sherpa --path=<prefix>/share/SHERPA-MC/Examples/V_plus_Jets/LHC_ZJets`. In the example, an absolute path is passed to the optional argument –path. It may also be specified relative to the current working directory. If it is not specified at all, the current working directory is understood.

For good book-keeping, it is highly recommended to reserve different subdirectories for different simulations as is demonstrated with the example setups.

If AMEGIC++ is used, Sherpa requires an initialization run, where C++ source code is written to disk. This code must be compiled into dynamic libraries by the user by running the makelibs script in the working directory. Alternatively, if scons is installed, you may invoke `<prefix>/bin/make2scons` and run `scons install`. After this step Sherpa is run again for the actual cross section integrations and event generation. For more information on and examples of how to run Sherpa using AMEGIC++, see *Running Sherpa with AMEGIC++*.

If the internal hard-coded matrix elements or Comix are used, and AMEGIC++ is not, an initialization run is not needed, and Sherpa will calculate the cross sections and generate events during the first run.

As the cross sections are integrated, the integration over phase space is optimized to arrive at an efficient event generation. Subsequently events are generated if a number of events is passed to the optional argument `--events` or set in the `Sherpa.yaml` file with the *EVENTS* parameters.

The generated events are not stored into a file by default; for details on how to store the events see *Event output formats*. Note that the computational effort to go through this procedure of generating, compiling and integrating the matrix elements of the hard processes depends on the complexity of the parton-level final states. For low multiplicities (2->2,3,4), of course, it can be followed instantly.

Usually more than one generation run is wanted. As long as the parameters that affect the matrix-element integration are not changed, it is advantageous to store the cross sections obtained during the generation run for later use. This saves CPU time especially for large final-state multiplicities of the matrix elements. Per default, Sherpa stores these integration results in a directory called `Results/`. The name of the output directory can be customised via *Results directory*

```
<prefix>/bin/Sherpa -r <result>/
```

or with `RESULT_DIRECTORY: <result>/` in the steering file, see *RESULT_DIRECTORY*. The storage of the integration results can be prevented by either using

```
<prefix>/bin/Sherpa -g
```

or by specifying `GENERATE_RESULT_DIRECTORY: false` in the steering file.

If physics parameters change, the cross sections have to be recomputed. The new results should either be stored in a new directory or the `<result>` directory may be re-used once it has been emptied. Parameters which require a recomputation are any parameters affecting the *Models*, *Matrix elements* or *Selectors*. Standard examples are changing the magnitude of couplings, renormalisation or factorisation scales, changing the PDF or centre-of-mass energy, or, applying different cuts at the parton level. If unsure whether a recomputation is required, a simple test is to temporarily use a different value for the `RESULT_DIRECTORY` option and check whether the new integration numbers (statistically) comply with the stored ones.

A warning on the validity of the process libraries is in order here: it is absolutely mandatory to generate new library files, whenever the physics model is altered, i.e. particles are added or removed and hence new or existing diagrams may or may not anymore contribute to the same final states. Also, when particle masses are switched on or off, new library files must be generated (however, masses may be changed between non-zero values keeping the same process libraries). The best recipe is to create a new and separate setup directory in such cases. Otherwise the `Process` and `Results` directories have to be erased:

```
$ rm -rf Process/ Results/
```

In either case one has to start over with the whole initialization procedure to prepare for the generation of events.

### 2.2.2 The example set-up: Z+Jets at the LHC

The setup file (`Sherpa.yaml`) provided in `./Examples/V_plus_Jets/LHC_ZJets/` can be considered as a standard example to illustrate the generation of fully hadronised events in Sherpa, cf. *Z+jets production*. Such events will include effects from parton showering, hadronisation into primary hadrons and their subsequent decays into stable hadrons. Moreover, the example chosen here nicely demonstrates how Sherpa is used in the context of merging matrix elements and parton showers [HKSS09]. In addition to the aforementioned corrections, this simulation of inclusive Drell-Yan production (electron-positron channel) will then include higher-order jet corrections at the tree level. As a result the transverse-momentum distribution of the Drell-Yan pair and the individual jet multiplicities as measured by the ATLAS and CMS collaborations at the LHC can be well described.

Before event generation, the initialization procedure as described in *Process selection and initialization* has to be completed. The matrix-element processes included in the setup are the following:

```
proton proton -> parton parton -> electron positron + up to four partons
```

In the PROCESSES list of the steering file this translates into .. code-block:: yaml

- **93 93 -> 90 90 93{4}:** Order: {QCD: Any, EW: 2} CKKW: 30

The physics model for these processes is the Standard Model (SM) which is the default setting of the parameter MODEL and is therefore not set explicitly. Fixing the order of electroweak couplings to 2, matrix elements of all partonic subprocesses for Drell-Yan production without any and with up to two extra QCD parton emissions will be generated. Proton–proton collisions are considered at beam energies of 3.5 TeV. The default PDF used by Sherpa is CT14. Model parameters and couplings can all be defined in the Sherpa.yaml file. Similarly, the way couplings are treated can be defined. As no options are set the default parameters and scale setting procedures are used.

The QCD radiation matrix elements have to be regularised to obtain meaningful cross sections. This is achieved by specifying CKKW: 30 when defining the process in Sherpa.yaml. Simultaneously, this tag initiates the ME-PS merging procedure. To eventually obtain fully hadronised events, the FRAGMENTATION setting has been left on it's default value Ahadic (and therefore been omitted from the steering file), which will run Sherpa's cluster hadronisation, and the DECAYMODEL setting has it's default value Hadrons, which will run Sherpa's hadron decays. Additionally corrections owing to photon emissions are taken into account.

To run this example set-up, use the

```
$ <prefix>/bin/Sherpa
```

command as described in *Running Sherpa*. Sherpa displays some output as it runs. At the start of the run, Sherpa initializes the relevant model, and displays a table of particles, with their *PDG codes* and some properties. It also displays the *Particle containers*, and their contents. The other relevant parts of Sherpa are initialized, including the matrix element generator(s). The Sherpa output will look like:

```
Welcome to Sherpa, <user name> on <host name>. Initialization of framework underway.
[...]
Random::SetSeed(): Seed set to 1234
[...]
Beam_Spectra_Handler :
   type = Monochromatic*Monochromatic
   for    P+  ((4000,0,0,4000))
   and    P+  ((4000,0,0,-4000))
PDF set 'ct14nn' loaded for beam 1 (P+).
PDF set 'ct14nn' loaded for beam 2 (P+).
Initialized the ISR.
Standard_Model::FixEWParameters() {
  Input scheme: 2
                alpha(m_Z) scheme, input: 1/\alphaQED(m_Z), m_W, m_Z, m_h, widths
  Ren. scheme:  2
                alpha(m_Z)
  Parameters:   sin^2(\theta_W) = 0.222928 - 0.00110708 i
                vev             = 243.034 - 3.75492 i
}
Running_AlphaQED::PrintSummary() {
  Setting \alpha according to EW scheme
  1/\alpha(0)   = 128.802
  1/\alpha(def) = 128.802
}
One_Running_AlphaS::PrintSummary() {
  Setting \alpha_s according to PDF
```

(continues on next page)

(continued from previous page)

```
  perturbative order 2
  \alpha_s(M_Z) = 0.118
}
[...]
Hadron_Init::Init(): Initializing kf table for hadrons.
Initialized the Fragmentation_Handler.
Initialized the Soft_Collision_Handler.
Initialized the Shower_Handler.
[...]
Matrix_Element_Handler::BuildProcesses(): Looking for processes .. done
Matrix_Element_Handler::InitializeProcesses(): Performing tests .. done
Matrix_Element_Handler::InitializeProcesses(): Initializing scales  done
Initialized the Matrix_Element_Handler for the hard processes.
Primordial_KPerp::Primordial_KPerp() {
  scheme = 0
  beam 1: P+, mean = 1.1, sigma = 0.914775
  beam 2: P+, mean = 1.1, sigma = 0.914775
}
Initialized the Beam_Remnant_Handler.
Hadron_Decay_Map::Read:   Initializing HadronDecays.dat. This may take some time.
Initialized the Hadron_Decay_Handler, Decay model = Hadrons
[...]
R
```

Then Sherpa will start to integrate the cross sections. The output will look like:

```
Process_Group::CalculateTotalXSec(): Calculate xs for '2_2__j__j__e-__e+' (Comix)
Starting the calculation at 11:58:56. Lean back and enjoy ... .
822.035 pb +- ( 16.9011 pb = 2.05601 % ) 5000 ( 11437 -> 43.7 % )
full optimization:  ( 0s elapsed / 22s left ) [11:58:56]
841.859 pb +- ( 11.6106 pb = 1.37916 % ) 10000 ( 18153 -> 74.4 % )
full optimization:  ( 0s elapsed / 21s left ) [11:58:57]
...
```

The first line here displays the process which is being calculated. In this example, the integration is for the 2->2 process, parton, parton -> electron, positron. The matrix element generator used is displayed after the process. As the integration progresses, summary lines are displayed, like the one shown above. The current estimate of the cross section is displayed, along with its statistical error estimate. The number of phase space points calculated is displayed after this (`10000` in this example), and the efficiency is displayed after that. On the line below, the time elapsed is shown, and an estimate of the total time till the optimisation is complete. In square brackets is an output of the system clock.

When the integration is complete, the output will look like:

```
...
852.77 pb +- ( 0.337249 pb = 0.0395475 % ) 300000 ( 313178 -> 98.8 % )
integration time:  ( 19s elapsed / 0s left ) [12:01:35]
852.636 pb +- ( 0.330831 pb = 0.038801 % ) 310000 ( 323289 -> 98.8 % )
integration time:  ( 19s elapsed / 0s left ) [12:01:35]
2_2__j__j__e-__e+ : 852.636 pb +- ( 0.330831 pb = 0.038801 % )  exp. eff: 13.4945 %
  reduce max for 2_2__j__j__e-__e+ to 0.607545 ( eps = 0.001 )
```

with the final cross section result and its statistical error displayed.

Sherpa will then move on to integrate the other processes specified in the run card.

When the integration is complete, the event generation will start. As the events are being generated, Sherpa will display a summary line stating how many events have been generated, and an estimate of how long it will take. When

the event generation is complete, Sherpa's output looks like:

```
Event 10000 ( 72 s total ) = 1.20418e+07 evts/day
In Event_Handler::Finish : Summarizing the run may take some time.
+----------------------------------------------------+
|                                                    |
|  Total XS is 900.147 pb +- ( 8.9259 pb = 0.99 % )  |
|                                                    |
+----------------------------------------------------+
```

A summary of the number of events generated is displayed, with the total cross section for the process.

The generated events are not stored into a file by default; for details on how to store the events see *Event output formats*.

### 2.2.3 Parton-level event generation with Sherpa

Sherpa has its own tree-level matrix-element generators called AMEGIC++ and Comix. Furthermore, with the module PHASIC++, sophisticated and robust tools for phase-space integration are provided. Therefore Sherpa obviously can be used as a cross-section integrator. Because of the way Monte Carlo integration is accomplished, this immediately allows for parton-level event generation. Taking the `LHC_ZJets` setup, users have to modify just a few settings in `Sherpa.yaml` and would arrive at a parton-level generation for the process gluon down-quark to electron positron and down-quark, to name an example. When, for instance, the options "`EVENTS: 0`" and "`OUTPUT: 2`" are added to the steering file, a pure cross-section integration for that process would be obtained with the results plus integration errors written to the screen.

For the example, the process definition in `PROCESSES` simplifies to

```
- 21 1 -> 11 -11 1:
    Order: {QCD: Any, EW: 2}
```

with all other settings in the process block removed. And under the assumption to start afresh, the initialization procedure has to be followed as before. Picking the same collider environment as in the previous example there are only two more changes before the `Sherpa.yaml` file is ready for the calculation of the hadronic cross section of the process g d to e- e+ d at LHC and subsequent parton-level event generation with Sherpa. These changes read `SHOWER_GENERATOR: None`, to switch off parton showering, `FRAGMENTATION: None`, to do so for the hadronisation effects, `MI_HANDLER: None`, to switch off multiparton interactions, and `ME_QED: {ENABLED: false}`, to switch off resummed QED corrections onto the $Z \to e^- e^+$ decay. Additionally, the non-perturbative intrinsic transverse momentum may be wished to not be taken into account, therefore set `BEAM_REMNANTS: false`.

### 2.2.4 Multijet merged event generation with Sherpa

For a large fraction of LHC final states, the application of reconstruction algorithms leads to the identification of several hard jets. Calculations therefore need to describe as accurately as possible both the hard jet production as well as the subsequent evolution and the interplay of multiple such topologies. Several scales determine the evolution of the event.

Various such merging schemes have been proposed: [CKKW01], [Lon02], [MMP02], [Kra02], [MMPT07], [LL08], [HKSS09], [HRT09], [HN10], [HKSS11], [LP12], [HKSS13], [GHK+13], [LPb], [LPa]. Comparisons of the older approaches can be found e.g. in [H+], [A+08]. The currently most advanced treatment at tree-level, detailed in [HKSS09], [HSS10], [CGH10], is implemented in Sherpa.

How to setup a multijet merged calculation is detailed in most *Examples*, eg. *W+jets production*, *Z+jets production* or *Top quark (pair) + jets production*.

### 2.2.5 Running Sherpa with AMEGIC++

When Sherpa is run using the matrix element generator AMEGIC++, it is necessary to run it twice. During the first run (the initialization run) Feynman diagrams for the hard processes are constructed and translated into helicity amplitudes. Furthermore suitable phase-space mappings are produced. The amplitudes and corresponding integration channels are written to disk as C++ source code, placed in a subdirectory called `Process`. The initialization run is started using the standard Sherpa executable, as described in *Running Sherpa*. The relevant command is

```
$ <prefix>/bin/Sherpa
```

The initialization run stops with the message "New libraries created. Please compile.", which is nothing but the request to carry out the compilation and linking procedure for the generated matrix-element libraries. The `makelibs` script, provided for this purpose and created in the working directory, must be invoked by the user (see `./makelibs -h` for help):

```
$ ./makelibs
```

Note that the following tools have to be available for this step: `autoconf`, `automake` and `libtool`.

Alternatively, if scons is installed, you may invoke `<prefix>/bin/make2scons` and run `scons install`. If scons was detected during the compilation of Sherpa, also makelibs uses `scons` per default (can be forced to use `autotools` by `./makelibs -s`).

Another option is `./makelibs -m`, which creates one library per subprocess. This can be useful for very complex processes, in particular if the default combined library generation fails due to a limit on the number of command line arguments. Note that this option requires that Sherpa is run with `AMEGIC_LIBRARY_MODE: 0` (default: 1).

Afterwards Sherpa can be restarted using the same command as before. In this run (the generation run) the cross sections of the hard processes are evaluated. Simultaneously the integration over phase space is optimized to arrive at an efficient event generation.

## 2.3 Cross section determination

To determine the total cross section, in particular in the context of multijet merging with Sherpa, the final output of the event generation run should be used, e.g.

```
+----------------------------------------------------+
|                                                    |
|  Total XS is 1612.17 pb +- ( 8.48908 pb = 0.52 % )  |
|                                                    |
+----------------------------------------------------+
```

Note that the Monte Carlo error quoted for the total cross section is determined during event generation. It, therefore, might differ substantially from the errors quoted during the integration step, and it can be reduced simply by generating more events.

In contrast to plain fixed order results, Sherpa's total cross section in multijet merging setups (MEPS, MENLOPS, MEPS@NLO) is composed of values from various fixed order processes, namely those which are combined by applying the multijet merging, see *Multijet merged event generation with Sherpa*. In this context, it is important to note:

**The higher multiplicity tree-level cross sections determined during the integration step are meaningless by themselves, only the inclusive cross section printed at the end of the event generation run is to be used.**

**Sherpa total cross sections have leading order accuracy when the generator is run in LO merging mode (MEPS), in NLO merging (MENLOPS, MEPS@NLO) mode they have NLO accuracy.**

### 2.3.1 Differential cross sections from single events

To calculate the expectation value of an observable defined through a series of cuts and requirements each event produced by Sherpa has to be evaluated whether it meets the required criteria. The expectation value is then given by

$$\langle O \rangle = \frac{1}{N_{\text{trial}}} \cdot \sum_i^n w_i(\Phi_i) O(\Phi_i).$$

Therein the $w_i(\Phi_i)$ are the weight of the event with the phase space configuration $\Phi_i$ and $O(\Phi_i)$ is the value of the observable at this point. $N_{\text{trial}} = \sum_i^n n_{\text{trial,i}}$ is the sum of number of trials $n_{\text{trial,i}}$ of all events. A good cross check is to reproduce the inclusive cross section as quoted by Sherpa (see above).

In case of unweighted events one might want to rescale the uniform event weight to unity using `w_norm`. The above equation then reads

$$\langle O \rangle = \frac{w_{\text{norm}}}{N_{\text{trial}}} \cdot \sum_i^n \frac{w_i(\Phi_i)}{w_{\text{norm}} O(\Phi_i)}$$

wherein $\frac{w_i(\Phi_i)}{w_norm} = 1$, i.e. the sum simply counts how many events pass the selection criteria of the observable. If however, `PartiallyUnweighted` event weights or `Enhance_Factor` or `Enhance_Observable` are used, this is no longer the case and the full form needs to be used.

All required quantities, $w_i$, $w_{\text{norm}}$ and $n_{\text{trial},i}$, accompany each event and are written e.g. into the HepMC output (cf. *Event output formats*).

CHAPTER

THREE

# COMMAND LINE OPTIONS

The available command line options for Sherpa.

**--run-data, -f** <file>
: Read settings from input file <file>, see *Input structure*. This is deprecated, use positional arguments to specify input files instead, see *Input structure*.

**--path, -p** <path>
: Read input file from path <path>, see *Input structure*.

**--sherpa-lib-path, -L** <path>
: Set Sherpa library path to <path>, see *SHERPA_CPP_PATH*.

**--events, -e** <N_events>
: Set number of events to generate <N_events>, see *EVENTS*.

**--event-type, -t** <event_type>
: Set the event type to <event_type>, see *EVENT_TYPE*.

**--result-directory, -r** <path>
: Set the result directory to <path>, see *RESULT_DIRECTORY*.

**--random-seed, -R** <seed>
: Set the seed of the random number generator to <seed>, see *RANDOM_SEED*.

**--me-generators, -m** <generators>
: Set the matrix element generator list to <generators>, see *ME_GENERATORS*. If you specify more than on generator, use the YAML sequence syntax, e.g. `-m '[Amegic, Comix]'`.

**--mi-handler, -M** <handler>
: Set multiple interaction handler to <handler>, see *MI_HANDLER*.

**--event-generation-mode, -w** <mode>
: Set the event generation mode to <mode>, see *EVENT_GENERATION_MODE*.

**--shower-generator, -s** <generator>
: Set the parton shower generator to <generator>, see *SHOWER_GENERATOR*.

**--fragmentation, -F** <module>
: Set the fragmentation module to <module>, see *Fragmentation*.

**--decay, -D** <module>
: Set the hadron decay module to <module>, see *Hadron decays*.

**--analysis, -a** <analyses>
: Set the analysis handler list to <analyses>, see *ANALYSIS*. If you specify more than one analysis handler, use the YAML sequence syntax, e.g. `-a '[Rivet, Internal]'`.

**--analysis-output, -A** <path>
Set the analysis output path to <path>, see *ANALYSIS_OUTPUT*.

**--output, -O** <level>
Set general output level <level>, see *OUTPUT*.

**--event-output, -o** <level>
Set output level for event generation <level>, see *OUTPUT*.

**--log-file, -l** <logfile>
Set log file name <logfile>, see *LOG_FILE*.

**--disable-result-directory-generation, -g**
Do not create result directory, see *RESULT_DIRECTORY*.

**--disable-batch-mode, -b**
Switch to non-batch mode, see *BATCH_MODE*.

**--print-version-info, -V**
Print extended version information at startup.

**--version, -v**
Print versioning information.

**--help, -h**
Print a help message.

**'PARAMETER:** Value'
Set the value of a parameter, see *Parameters*.

**'Tags:** {TAG: Value}'
Set the value of a tag, see *Tags*.

CHAPTER

FOUR

# INPUT STRUCTURE

A Sherpa setup is steered by various parameters, associated with the different components of event generation.

These have to be specified in a configuration file which by default is named `Sherpa.yaml` in the current working directory. If you want to use a different setup directory for your Sherpa run, you have to specify it on the command line as `-p <dir>` or `'PATH: <dir>'` (including the quotes).

To read parameters from a configuration file with a different name, you may give the file name as a positional argument on the command line like this: `Sherpa <file>`. Note that you can also pass more than one file like this: `Sherpa <file1> <file2> ...` In this case, settings in files to the right take precedence. This can be useful to reduce duplication in the case that you have several setups that share a common set of settings.

Note that you can also pass filenames using the legacy syntax `-f <file>` or `'RUNDATA: [<file1>, <file2>]'`. However, this is deprecated. Use positional arguments instead. Mixing this legacy syntax and positional arguments for specifying configuration files is undefined behaviour.

Sherpa's configuration files are written in the YAML format. Most settings are just written as the settings' name followed by its value, like this:

```
EVENTS: 100M
BEAMS: 2212
BEAM_ENERGIES: 7000
...
```

Others use a more nested structure:

```
HARD_DECAYS:
    Enabled: true
    Apply_Branching_Ratios: false
```

where `Enabled` and `Apply_Branching_Ratios` are sub-settings of the top-level `HARD_DECAYS` setting, which is denoted by indentation (here two additional spaces).

The different settings and their structure are described in detail in another chapter of this manual, see *Parameters*.

All parameters can be overwritten on the command line, i.e. command-line input has the highest priority. Each argument is parsed as a single YAML line. This usually means that you have to quote each argument:

```
$ <prefix>/bin/Sherpa 'KEYWORD1: value1' 'KEYWORD2: value2' ...
```

Because each argument is parsed as YAML, you can also specify nested settings, e.g. to disable hard decays (even if it is enabled in the config file) you can write:

```
$ <prefix>/bin/Sherpa 'HARD_DECAYS: {Enabled: false}'
```

Or you can specify the list of matrix-element generators writing:

```
$ <prefix>/bin/Sherpa 'ME_GENERATORS: [Comix, Amegic]'
```

All over Sherpa, particles are defined by the particle code proposed by the PDG. These codes and the particle properties will be listed during each run with `OUTPUT: 2` for the elementary particles and `OUTPUT: 4` for the hadrons. In both cases, antiparticles are characterized by a minus sign in front of their code, e.g. a mu- has code `13`, while a mu+ has `-13`.

All quantities have to be specified in units of GeV and millimeter. The same units apply to all numbers in the event output (momenta, vertex positions). Scattering cross sections are denoted in pico-barn in the output.

There are a few extra features for an easier handling of the parameter file(s), namely global tag replacement, see *Tags*, and algebra interpretation, see *Interpreter*.



## 4.1 Interpreter

Sherpa has a built-in interpreter for algebraic expressions, like `cos(5/180*M_PI)`. This interpreter is employed when reading integer and floating point numbers from input files, such that certain parameters can be written in a more convenient fashion. For example it is possible to specify the factorisation scale as `sqr(91.188)`.

There are predefined tags to alleviate the handling

**`M_PI`** Ludolph's Number to a precision of 12 digits.

**`M_C`** The speed of light in the vacuum.

**`E_CMS`** The total centre of mass energy of the collision.

The expression syntax is in general C-like, except for the extra function `sqr`, which gives the square of its argument. Operator precedence is the same as in C. The interpreter can handle functions with an arbitrary list of parameters, such as `min` and `max`.

The interpreter can be employed to construct arbitrary variables from four momenta, like e.g. in the context of a parton level selector, see *Selectors*. The corresponding functions are

**`Mass(v)`** The invariant mass of $v$ in GeV.

**`Abs2(v)`** The invariant mass squared of $v$ in GeV^2.

**`PPerp(v)`** The transverse momentum of $v$ in GeV.

**`PPerp2(v)`** The transverse momentum squared of $v$ in GeV^2.

**`MPerp(v)`** The transverse mass of $v$ in GeV.

**`MPerp2(v)`** The transverse mass squared of $v$ in GeV^2.

**`Theta(v)`** The polar angle of $v$ in radians.

**`Eta(v)`** The pseudorapidity of $v$.

**`Y(v)`** The rapidity of $v$.

**`Phi(v)`** The azimuthal angle of $v$ in radians.

**`Comp(v,i)`** **The *i*'th component of the vector** $v$. $i$ = 0 is the energy/time component, $i$ = 1, 2, and 3 are the x, y, and z components.

**PPerpR(*v1*,*v2*)** The relative transverse momentum between *v1* and *v2* in GeV.

**ThetaR(*v1*,*v2*)** The relative angle between *v1* and *v2* in radians.

**DEta(*v1*,*v2*)** The pseudo-rapidity difference between *v1* and *v2*.

**DY(*v1*,*v2*)** The rapidity difference between *v1* and *v2*.

**DPhi(*v1*,*v2*)** The relative polar angle between *v1* and *v2* in radians.

## 4.2 Tags

Tag replacement in Sherpa is performed through the data reading routines, which means that it can be performed for virtually all inputs. Specifying a tag on the command line or in the configuration file using the syntax `TAGS: {<Tag>: <Value>}` will replace every occurrence of `<Tag>` in all files during read-in. An example tag definition could read

```
$ <prefix>/bin/Sherpa 'TAGS: {QCUT: 20, NJET: 3}'
```

and then be used in the configuration file like:

```
RESULT_DIRECTORY: Result_$(QCUT)
PROCESSES:
- 93 93 -> 11 -11 93{$(NJET)}:
    Order: {QCD: Any, EW: 2}
    CKKW: $(QCUT)
```

CHAPTER

FIVE

# PARAMETERS

A Sherpa run is steered by various parameters, associated with the different components of event generation. These are set in Sherpa's configuration file, see *Input structure* for more details. Tag replacements may be performed in all inputs, see *Tags*.

## 5.1 General parameters

The following parameters describe general run information. See *Input structure* for how to use them in a configuration file or on the command line.

### 5.1.1 EVENTS

This parameter specifies the number of events to be generated.

It can alternatively be set on the command line through option `-e`, see *Command Line Options*.

### 5.1.2 EVENT_TYPE

This parameter specifies the kind of events to be generated. It can alternatively be set on the command line through option `-t`, see *Command Line Options*.

- The default event type is `StandardPerturbative`, which will generate a hard event through exact matrix elements matched and/or merged with the paerton shower, eventually including hadronization, hadron decays, etc..

Alternatively there are two more specialised modes, namely:

- `MinimumBias`, which generates minimum bias events through the SHRIMPS model implemented in Sherpa, see *Minimum bias events*
- `HadronDecay`, which allows to simulate the decays of a specific hadron.

### 5.1.3 SHERPA_VERSION

This parameter ties a config file to a specific Sherpa version, e.g. `SHERPA_VERSION: 2.2.0`. If two parameters are given they are interpreted as a range of Sherpa versions: `SHERPA_VERSION: [2.2.0, 2.2.5]` specifies that this config file can be used with any Sherpa version between (and including) 2.2.0 and 2.2.5.

### 5.1.4 TUNE

**Warning:** This parameter is currently not supported.

### 5.1.5 OUTPUT

This parameter specifies the screen output level (verbosity) of the program. If you are looking for event file output options please refer to section *Event output formats*.

It can alternatively be set on the command line through option `-O`, see *Command Line Options*. A different output level can be specified for the event generation step through `EVT_OUTPUT` or command line option `-o`, see *Command Line Options*

The value can be any sum of the following:

- 0: Error messages (-> always displayed).
- 1: Event display.
- 2: Informational messages during the run.
- 4: Tracking messages (lots of output).
- 8: Debugging messages (even more output).

E.g. `OUTPUT=3` would display information, events and errors. Use `OUTPUT_PRECISION` to set the default output precision (default `6`). Note: this may be overriden in specific functions' output.

For expert users: The output level can be overriden for individual functions, e.g. like this

```
FUNCTION_OUTPUT:
  "void SHERPA::Matrix_Element_Handler::BuildProcesses()": 8
  ...
```

where the function signature is given by the value of `__PRETTY_FUNCTION__` in the function block. Another expert parameter is `EVT_OUTPUT_START`, with which the first event affected by `EVT_OUTPUT` can be specified. This can be useful to generate debugging output only for events affected by a some issue.

### 5.1.6 LOG_FILE

This parameter specifies the log file. If set, the standard output from Sherpa is written to the specified file, but output from child processes is not redirected. This option is particularly useful to produce clean log files when running the code in MPI mode, see *MPI parallelization*. A file name can alternatively be specified on the command line through option `-l`, see *Command Line Options*.

### 5.1.7 RANDOM_SEED

Sherpa uses different random-number generators. The default is the Ran3 generator described in [PTVF07]. Alternatively, a combination of George Marsaglias KISS and SWB [MZ91] can be employed, see this website. The integer-valued seeds of the generators are specified by `RANDOM_SEED: [A, .., D]`. They can also be set individually using `RANDOM_SEED1: A` through `RANDOM_SEED4: D`. The Ran3 generator takes only one argument (in this case, you can simply use `RANDOM_SEED: A`). This value can also be set using the command line option `-R`, see *Command Line Options*.

### 5.1.8 EVENT_SEED_MODE

The tag `EVENT_SEED_MODE` can be used to enforce the same seeds in different runs of the generator. When set to 1, existing random seed files are read and the seed is set to the next available value in the file before each event. When set to 2, seed files are written to disk. These files are gzip compressed, if Sherpa was compiled with option `--enable-gzip`. When set to 3, Sherpa uses an internal bookkeeping mechanism to advance to the next predefined seed. No seed files are written out or read in.

### 5.1.9 ANALYSIS

Analysis routines can be switched on or off using the ANALYSIS parameter. The default is no analysis. This parameter can also be specified on the command line using option `-a`, see *Command Line Options*.

The following analysis handlers are currently available

**`Internal`**

Sherpa's internal analysis handler.

To use this option, the package must be configured with option `--enable-analysis`.

An output directory can be specified using *ANALYSIS_OUTPUT*.

**`Rivet`**

The Rivet package, see Rivet Website.

To enable it, Rivet and HepMC have to be installed and Sherpa must be configured as described in *Rivet analyses*.

**HZTool**

The HZTool package, see HZTool Website.
To enable it, HZTool and CERNLIB have to be installed and Sherpa must be configured as described in *HZTool analyses*.

Multiple options can also be specified, e.g. `ANALYSIS: [Internal, Rivet]`.

### 5.1.10 ANALYSIS_OUTPUT

Name of the directory for histogram files when using the internal analysis and name of the Yoda file when using Rivet, see *ANALYSIS*. The directory/file will be created w.r.t. the working directory. The default value is `Analysis/`. This parameter can also be specified on the command line using option `-A`, see *Command Line Options*.

### 5.1.11 TIMEOUT

A run time limitation can be given in user CPU seconds through `TIMEOUT`. This option is of some relevance when running SHERPA on a batch system. Since in many cases jobs are just terminated, this allows to interrupt a run, to store all relevant information and to restart it without any loss. This is particularly useful when carrying out long integrations. Alternatively, setting the `TIMEOUT` variable to -1, which is the default setting, translates into having no run time limitation at all. The unit is seconds.

### 5.1.12 RLIMIT_AS

A memory limitation can be given to prevent Sherpa to crash the system it is running on as it continues to build up matrix elements and loads additional libraries at run time. Per default the maximum RAM of the system is determined and set as the memory limit. This can be changed by giving `RLIMIT_AS:` where the size is given as e.g. `500 MB`, `4 GB`, or `10 %`. When running with *MPI parallelization* it might be necessary to divide the total maximum by the number of cores. This can be done by setting `RLIMIT_BY_CPU: true`.

Sherpa checks for memory leaks during integration and event generation. If the allocated memory after start of integration or event generation exceeds the parameter `MEMLEAK_WARNING_THRESHOLD`, a warning is printed. Like `RLIMIT_AS`, `MEMLEAK_WARNING_THRESHOLD` can be set using units. The warning threshold defaults to `16MB`.

### 5.1.13 BATCH_MODE

Whether or not to run Sherpa in batch mode. The default is `1`, meaning Sherpa does not attempt to save runtime information when catching a signal or an exception. On the contrary, if option `0` is used, Sherpa will store potential integration information and analysis results, once the run is terminated abnormally. All possible settings are:

***0*** Sherpa attempts to write out integration and analysis results when catching an exception.

***1*** Sherpa does not attempt to write out integration and analysis results when catching an exception.

***2*** Sherpa outputs the event counter continously, instead of overwriting the previous one (default when using *LOG_FILE*).

***4*** Sherpa increases the on-screen event counter in constant steps of 100 instead of an increase relative to the current event number. The interval length can be adjusted with `EVENT_DISPLAY_INTERVAL`.

The settings are additive such that multiple settings can be employed at the same time.

---

**Note:** When running the code on a cluster or in a grid environment, BATCH_MODE should always contain setting 1 (i.e. `BATCH_MODE=[1|3|5|7]`).

The command line option `-b` should therefore not be used in this case, see *Command Line Options*.

---

### 5.1.14 NUM_ACCURACY

The targeted numerical accuracy can be specified through `NUM_ACCURACY`, e.g. for comparing two numbers. This might have to be reduced if gauge tests fail for numerical reasons. The default is `1E-10`.

### 5.1.15 SHERPA_CPP_PATH

The path in which Sherpa will eventually store dynamically created C++ source code. If not specified otherwise, sets `SHERPA_LIB_PATH` to `$SHERPA_CPP_PATH/Process/lib`. This value can also be set using the command line option `-L`, see *Command Line Options*. Both settings can also be set using environment variables.

### 5.1.16 SHERPA_LIB_PATH

The path in which Sherpa looks for dynamically linked libraries from previously created C++ source code, cf. *SHERPA_CPP_PATH*.

### 5.1.17 Event output formats

Sherpa provides the possibility to output events in various formats, e.g. the HepEVT common block structure or the HepMC format. The authors of Sherpa assume that the user is sufficiently acquainted with these formats when selecting them.

If the events are to be written to file, the parameter `EVENT_OUTPUT` must be specified together with a file name. An example would be `EVENT_OUTPUT: HepMC_GenEvent[MyFile]`, where `MyFile` stands for the desired file base name. More than one output can also be specified:

```
EVENT_OUTPUT:
  - HepMC_GenEvent[MyFile]
  - Root[MyFile]
```

The following formats are currently available:

**HepMC_GenEvent** Generates output in HepMC::IO_GenEvent format. The HepMC::GenEvent::m_weights weight vector stores the following items: `[0]` event weight, `[1]` combined matrix element and PDF weight (missing only phase space weight information, thus directly suitable for evaluating the matrix element value of the given configuration), `[2]` event weight normalisation (in case of unweighted events event weights of ~ +/-1 can be obtained by (event weight)/(event weight normalisation)), and `[3]` number of trials. The total cross section of the simulated event sample can be computed as the sum of event weights divided by the sum of the number of trials. This value must agree with the total cross section quoted by Sherpa at the end of the event generation run, and it can serve as a cross-check on the consistency of the HepMC event file. Note that Sherpa conforms to the Les Houches 2013 suggestion (http://phystev.in2p3.fr/wiki/2013:groups:tools:hepmc) of indicating interaction types through the GenVertex type-flag. Multiple event weights can also be enabled with HepMC versions >=2.06, cf. *Scale and PDF variations*. The following additional customisations can be used

HEPMC_USE_NAMED_WEIGHTS: <false|true> Enable filling weights with an associated name. The nominal event weight has the key Weight. MEWeight, WeightNormalisation and NTrials provide additional information for each event as described above. Needs HepMC version >=2.06.

HEPMC_EXTENDED_WEIGHTS: <false|true> Write additional event weight information needed for a posteriori reweighting into the WeightContainer, cf. *A posteriori scale and PDF variations using the HepMC GenEvent Output*. Necessitates the use of HEPMC_USE_NAMED_WEIGHTS.

HEPMC_TREE_LIKE: <false|true> Force the event record to be stricly tree-like. Please note that this removes some information from the matrix-element-parton-shower interplay which would be otherwise stored.

HepMC_Short

Generates output in HepMC::IO_GenEvent format, however, only incoming beams and outgoing particles are stored. Intermediate and decayed particles are not listed. The event weights stored as the same as above, and HEPMC_USE_NAMED_WEIGHTS and HEPMC_EXTENDED_WEIGHTS can be used to customise.

**HepMC3_GenEvent** Generates output using HepMC3 library. The format of the output is set with HEPMC3_IO_TYPE: <0|1|2|3|4> tag. The default value is 0 and corresponds to ASCII GenEvent. Other available options are 1: HepEvt 2: ROOT file with every event written as an object of class GenEvent. 3: ROOT file with GenEvent objects writen into TTree. Otherwise similar to HepMC_GenEvent.

**Delphes_GenEvent** Generates output in Root format, which can be passed to Delphes for analyses. Input events are taken from the HepMC interface. Storage space can be reduced by up to 50% compared to gzip compressed HepMC. This output format is available only if Sherpa was configured and installed with options --enable-root and --enable-delphes=/path/to/delphes.

**Delphes_Short** Generates output in Root format, which can be passed to Delphes for analyses. Only incoming beams and outgoing particles are stored.

**PGS** Generates output in StdHEP format, which can be passed to PGS for analyses. This output format is available only if Sherpa was configured and installed with options --enable-hepevtsize=4000 and --enable-pgs=/path/to/pgs. Please refer to the PGS documentation for how to pass StdHEP event files on to PGS. If you are using the LHC olympics executeable, you may run ./olympics --stdhep events.lhe <other options>.

**PGS_Weighted** Generates output in StdHEP format, which can be passed to PGS for analyses. Event weights in the HEPEV4 common block are stored in the event file.

**HEPEVT** Generates output in HepEvt format.

**LHEF** Generates output in Les Houches Event File format. This output format is intended for output of **matrix element configurations only**. Since the format requires PDF information to be written out in the outdated PDFLIB/LHAGLUE enumeration format this is only available automatically if LHAPDF is used, the identification numbers otherwise have to be given explicitly via LHEF_PDF_NUMBER (LHEF_PDF_NUMBER_1 and LHEF_PDF_NUMBER_2 if both beams carry different structure functions). This format currently outputs matrix element information only, no information about the large-Nc colour flow is given as the LHEF output format is not suited to communicate enough information for meaningful parton showering on top of multiparton final states.

**Root** Generates output in ROOT ntuple format **for NLO event generation only**. For details on the ntuple format, see *A posteriori scale and PDF variations using the ROOT NTuple Output*. This output option is available only if Sherpa was linked to ROOT during installation by using the configure option --enable-root=/path/to/root. ROOT ntuples can be read back into Sherpa and analyzed using the option EVENT_INPUT. This feature is described in *Production of NTuples*.

The output can be further customized using the following options:

**FILE_SIZE** Number of events per file (default: unlimited).

**EVENT_FILE_PATH** Directory where the files will be stored.

**EVENT_OUTPUT_PRECISION** Steers the precision of all numbers written to file (default: 12).

For all output formats except ROOT and Delphes, events can be written directly to gzipped files instead of plain text. The option `--enable-gzip` must be given during installation to enable this feature.

### 5.1.18 Scale and PDF variations

Sherpa can compute alternative event weights for different scale, PDF and AlphaS(MZ) choices on-the-fly, resulting in alternative weights for the generated event. This can be evoked with the following syntax:

```
VARIATIONS:
- ScaleFactors:
    MuR2: <muR2-fac-1>
    MuF2: <muF2-fac-1>
    QCUT: <qcut-fac-1>
  PDF: <PDF-1>
- ScaleFactors:
    MuR2: <muR2-fac-2>
    MuF2: <muF2-fac-2>
    QCUT: <qcut-fac-2>
  PDF: <PDF-2>
...
```

The key word `VARIATIONS` takes a list of variations. Each variation is specified by a set of scale factors, and a PDF choice (or AlphaS(MZ) choice, see below).

Scale factors can be given for the renormalisation, factorisation and for the merging scale. The corresponding keys are `MuR2`, `MuF2` and `QCUT`, respectively. The factors for the renormalisation and factorisation scales must be given in their quadratic form, i.e. `MuR2: 4.0` applies a scale factor of 2.0 to the renormalisation scale. All scale factors can be omitted (they default to 1.0). Instead of `MuR2` and `MuF2`, one can also use the keyword `Mu2`. In this case, the given factor is applied to both the renormalisation and the factorisation scale.

For the `PDF` specification, any set present in any of the PDF library interfaces loaded through `PDF_LIBRARY` can be used. If no PDF set is given it defaults to the nominal one. Specific PDF members can be specified by appending the PDF set name with `/<member-id>`.

Instead of using `PDF: <PDF>` (which consistently also varies the strong coupling if the PDF has a different specification of it!), one can also specify a pure AlphaS variation by giving its value at the Z mass scale: `AlphaS(MZ): <alphas(mz)-value>`. This can be useful e.g. for leptonic productions.

It can be painful to write every variation explicitly, e.g. for 7-point scale factor variations or if one want variations for all members of a PDF set. Therefore an asterisk can be appended to some values, which results in an *expansion*. For PDF sets, this means that the variation is repeated for each member of that set. For scale factors, `4.0*` is expanded to itself, unity, and its inverse: `1.0/4.0, 1.0, 4.0`. A special meaning is reserved for specifying `Mu2: 4.0*`, which expands to a 7-point scale variation:

```
VARIATIONS:
  - ScaleFactors:
      Mu2: 4.0*
```

is therefore equivalent to

```
VARIATIONS:
  - ScaleFactors:
      MuF2: 0.25
```

(continues on next page)

(continued from previous page)

```
      MuR2: 0.25
  - ScaleFactors:
      MuF2: 1.0
      MuR2: 0.25
  - ScaleFactors:
      MuF2: 0.25
      MuR2: 1.0
  - ScaleFactors:
      MuF2: 1.0
      MuR2: 1.0
  - ScaleFactors:
      MuF2: 4.0
      MuR2: 1.0
  - ScaleFactors:
      MuF2: 1.0
      MuR2: 4.0
  - ScaleFactors:
      MuF2: 4.0
      MuR2: 4.0
```

As another example, a complete variation using the PDF4LHC convention would read

```
VARIATIONS:
  - ScaleFactors:
      Mu2: 4.0*
  - PDF: CT10nlo*
  - PDF: MMHT2014nlo68cl*
  - PDF: NNPDF30_nlo_as_0118*
```

Please note, this syntax will create $7+53+51+101 = 212$ additional weights for each event. Even though reweighting is used to reduce the amount of additional calculation as far as possible, this can still necessitate a considerable amount of additional CPU hours, in particular when parton-shower reweighting is enabled (see below).

Note that asterisk expansions include trivial scale variations and the central PDF set. Depending on the other specifications in a variation, this could result in a completely trivial variation. Per default, these are omitted during the calculation, since the nominal calculation is anyway included in the Sherpa output. Trivial variations can be explicitly allowed using `VARIATIONS_INCLUDE_CV: false`.

The additional event weights can then be written into the event output. However, this is currently only supported for `HepMC_GenEvent` and `HepMC_Short` with versions >=2.06 and `HEPMC_USE_NAMED_WEIGHTS: true`. The alternative event weights follow the Les Houches naming convention for such variations, i.e. they are named `MUR<fac>_MUF<fac>_PDF<id>`. When using Sherpa's interface to Rivet, *Rivet analyses*, separate instances of Rivet, one for each alternative event weight in addition to the nominal one, are instantiated leading to one set of histograms each. They are again named using the `MUR<fac>_MUF<fac>_PDF<id>` convention. Extending this convention, for pure strong coupling variations, an additional tag `ASMZ<val>` is appended. Another set of tags is appended if shower scale variations are enabled, then giving `PSMUR<fac>_PSMUF<fac>`.

The user must also be aware that, of course, the cross section of the event sample, changes when using an alternative event weight as compared to the nominal one. Any histogramming therefore has to account for this and recompute the total cross section as the sum of weights divided by the number of trials, cf. *Cross section determination*.

The on-the-fly reweighting works for all event generation modes (weighted or (partially) unweighted) and all calculation types (LO, LOPS, NLO, NLOPS, MEPS@LO, MEPS@NLO and MENLOPS). However, the reweighting of parton shower emissions has to be enabled explicitly, using `CSS_REWEIGHT: true`. This should work out of the box for all types of variations. However, parton-shower reweighting (even though formally exact), tends to be numerically less stable than the reweighting of the hard process. If numerical issues are encountered, one can try to increase `CSS_REWEIGHT_SCALE_CUTOFF` (default: 5, measured in GeV). This disables shower variations for emissions at

scales below the value. An additional safeguard against rare spuriously large shower variation weights is implemented as `CSS_MAX_REWEIGHT_FACTOR` (default: 1e3). Any variation weights accumulated during an event and larger than this factor will be ignored and reset to 1.

To include the ME-only variations along with the full variations in the HepMC/Rivet output, you can use `HEPMC_INCLUDE_ME_ONLY_VARIATIONS: true` and `RIVET: { INCLUDE_HEPMC_ME_ONLY_VARIATIONS: true }`, respectively.

### 5.1.19 MPI parallelization

MPI parallelization in Sherpa can be enabled using the configuration option `--enable-mpi`. Sherpa supports OpenMPI and MPICH2 . For detailed instructions on how to run a parallel program, please refer to the documentation of your local cluster resources or the many excellent introductions on the internet. MPI parallelization is mainly intended to speed up the integration process, as event generation can be parallelized trivially by starting multiple instances of Sherpa with different random seed, cf. *RANDOM_SEED*. However, both the internal analysis module and the Root NTuple writeout can be used with MPI. Note that these require substantial data transfer.

Please note that the process information contained in the `Process` directory for both Amegic and Comix needs to be generated without MPI parallelization first. Therefore, first run

```
$ Sherpa INIT_ONLY=1 <Sherpa.yaml>
```

and, in case of using Amegic, compile the libraries. Then start your parallized integration, e.g.

```
$ mpirun -n <n> Sherpa -e 0 <Sherpa.yaml>
```

After the integration has finished, you can submit individual jobs to generate event samples (with a different random seed for each job). Upon completion, the results can be merged.

## 5.2 Beam parameters

Mandatory settings to set up the colliding particle beams are

- The initial beam particles specified through `BEAMS`, given by their PDG particle number. For (anti)protons and (positrons) electrons, for example, these are given by $(-)2212$ or $(-)11$, respectively. The code for photons is 22. If you provide a single particle number, both beams will consist of that particle type. If the beams consist of different particles, a list of two values have to be provided.
- The energies of both incoming beams are defined through `BEAM_ENERGIES`, given in units of GeV. Again, single values apply to both beams, whereas a list of two values have to be given when the two beams do not have the same energy.

Examples would be

```
# LHC
BEAMS: 2212
BEAM_ENERGIES: 7000

# HERA
BEAMS: [-11, 2212]
BEAM_ENERGIES: [27.5, 820]
```

More options related to beamstrahlung and intrinsic transverse momentum can be found in the following subsections.

### 5.2.1 Beam Spectra

If desired, you can also specify spectra for beamstrahlung through `BEAM_SPECTRA`. The possible values are

**`Monochromatic`** The beam energy is unaltered and the beam particles remain unchanged. That is the default and corresponds to ordinary hadron-hadron or lepton-lepton collisions.

**`Laser_Backscattering`** This can be used to describe the backscattering of a laser beam off initial leptons. The energy distribution of the emerging photon beams is modelled by the CompAZ parametrization, see [Zar03]. Note that this parametrization is valid only for the proposed TESLA photon collider, as various assumptions about the laser parameters and the initial lepton beam energy have been made. See details below.

**`Simple_Compton`** This corresponds to a simple light backscattering off the initial lepton beam and produces initial-state photons with a corresponding energy spectrum. See details below.

**`EPA`** This enables the equivalent photon approximation for colliding protons, see [AGH+08]. The resulting beam particles are photons that follow a dipole form factor parametrization, cf. [BGMS74]. The authors would like to thank T. Pierzchala for his help in implementing and testing the corresponding code. See details below.

**`Spectrum_Reader`** A user defined spectrum is used to describe the energy spectrum of the assumed new beam particles. The name of the corresponding spectrum file needs to be given through the keywords `SPECTRUM_FILES`.

The `BEAM_SMIN` and `BEAM_SMAX` parameters may be used to specify the minimum/maximum fraction of cms energy squared after Beamstrahlung. The reference value is the total centre of mass energy squared of the collision, *not* the centre of mass energy after eventual Beamstrahlung.

The parameter can be specified using the internal interpreter, see *Interpreter*, e.g. as `BEAM_SMIN: sqr(20/E_CMS)`.

#### Laser Backscattering

The energy distribution of the photon beams is modelled by the CompAZ parametrisation, see [Zar03], with various assumptions valid only for the proposed TESLA photon collider. The laser energies can be set by `E_LASER`. `P_LASER` sets their polarisations, defaulting to `0.`. Both settings can either be set to a single value, applying to both beams, or to a list of two values, one for each beam. The `LASER_MODE` takes the values `-1`, `0`, and `1`, defaulting to `0`. `LASER_ANGLES` and `LASER_NONLINEARITY` can be set to `true` or to `false` (default).

#### Simple Compton

This corresponds to a simple light backscattering off the initial lepton beam and produces initial-state photons with a corresponding energy spectrum. It is a special case of the above Laser Backscattering with `LASER_MODE: -1`.

#### EPA

The equivalent photon approximation, cf. [AGH+08], [BGMS74], has a few free parameters:

**`EPA_q2Max`** Parameter of the EPA spectra of the two beams, defaults to `2.` in units of GeV squared.

**`EPA_ptMin`** Infrared regulator to the EPA beam spectra. Given in GeV, the value must be between `0.` and `1.` for EPA approximation to hold. Defaults to `0.`, i.e. the spectrum has to be regulated by cuts on the observable, cf *Selectors*.

**`EPA_Form_Factor`** Form factor model to be used on the beams. The options are `0` (pointlike), `1` (homogeneously charged sphere, `2` (gaussian shaped nucleus), and `3` (homogeneously charged sphere, smoothed at low and high x). Applicable only to heavy ion beams. Defaults to `0`.

**`EPA_AlphaQED`** Value of alphaQED to be used in the EPA. Defaults to `0.0072992701`.

`EPA_q2Max`, `EPA_ptMin`, `EPA_Form_Factor` can either be set to single values that are then applied to both beams, or to a list of two values, for the respective beams.

### 5.2.2 Intrinsic Transverse Momentum

The intrinsic transverse momentum of the colliding particles can be set by subsettings of the `INTRINSIC_KPERP` setting:

```
INTRINSIC_KPERP:
  Parameter_1: <value_1>
  Parameter_2: <value_2>
...
```

The possible parameters and their defaults are

**`ENABLED (default: true)`** Setting this to `false` disables the intrinsic transverse momentum altogether.

**`SCHEME (default for protons: 0)`** This parameter specifies the scheme to calculate the intrinsic transverse momentum of the beams in case of hadronic beams, such as protons.

**`MEAN (default: 1.1)`** This parameter specifies the mean intrinsic transverse momentum in GeV for the beams in case of hadronic beams, such as protons.

If two values are provided, the intrinsic momenta means of the two beams are set to these two values, respectively.

**`SIGMA (default: 0.85)`** This parameter specifies the width of the Gaussian distribution in GeV of the intrinsic transverse momenta for the beams in case of hadronic beams, such as protons.

If two values are provided, the intrinsic momenta widths of the two beams are set to these two values, respectively.

**`EXP (default: 0.55)`** This parameter specifies the energy extrapolation exponent of the width of the Gaussian distribution of intrinsic transverse momentum for the beams in case of hadronic beams, such as protons.

If two values are provided, the exponents for each of the two beams are set to these two values, respectively.

**EREF (default: 7000)** This parameter specifies the reference scale in GeV in the energy extrapolation of the width of the Gaussian distribution of intrinsic transverse momentum for the beams in case of hadronic beams, such as protons.

If two values are provided, the reference scales for each of the two beams are set to these two values, respectively.

If the option `BEAM_REMNANTS: false` is specified, pure parton-level events are simulated, i.e. no beam remnants are generated. Accordingly, partons entering the hard scattering process do not acquire primordial transverse momentum.

## 5.3 ISR parameters

The following parameters are used to steer the setup of beam substructure and initial state radiation (ISR).

**BUNCHES** Specify the PDG ID of the first (left) and second (right) bunch particle (or both if only one value is provided), i.e. the particle after eventual Beamstrahlung specified through the beam parameters, see *Beam parameters*. Per default these are taken to be identical to the values set using `BEAMS`, assuming the default beam spectrum is Monochromatic. In case the Simple Compton or Laser Backscattering spectra are enabled the bunch particles would have to be set to 22, the PDG code of the photon.

`ISR_SMIN/ISR_SMAX`

This parameter specifies the minimum fraction of cms energy squared after ISR. The reference value is the total centre of mass energy squared of the collision, *not* the centre of mass energy after eventual Beamstrahlung.

The parameter can be specified using the internal interpreter, see *Interpreter*, e.g. as `ISR_SMIN: sqr(20/E_CMS)`.

Sherpa provides access to a variety of structure functions. They can be configured with the following parameters.

**PDF_LIBRARY** This parameter takes the list of PDF interfaces to load. The following options are distributed with Sherpa:

**LHAPDFSherpa** Use PDF's from LHAPDF [WBG]. The interface is only available if Sherpa has been compiled with support for LHAPDF, see *Installation*.

**CT14Sherpa** Built-in library for some PDF sets from the CTEQ collaboration, cf. [D+]. This is the default.

**CT12Sherpa** Built-in library for some PDF sets from the CTEQ collaboration, cf. [GGH+].

**CT10Sherpa** Built-in library for some PDF sets from the CTEQ collaboration, cf. [LGH+10].

**CTEQ6Sherpa** Built-in library for some PDF sets from the CTEQ collaboration, cf. [N+08].

**NNPDF30NLO** Built-in library for PDF sets from the NNPDF group, cf. [B+].

**MSTW08Sherpa** Built-in library for PDF sets from the MSTW group, cf. [MSTW09].

**MRST04QEDSherpa** Built-in library for photon PDF sets from the MRST group, cf. [MRST05].

**MRST01LOSherpa** Built-in library for the 2001 leading-order PDF set from the MRST group, cf. [MRST02].

**MRST99Sherpa** Built-in library for the 1999 PDF sets from the MRST group, cf. [MRST00].

**GRVSherpa** Built-in library for the GRV photon PDF [GRV92b], [GRV92a]

**PDFESherpa** Built-in library for the electron structure function. The perturbative order of the fine structure constant can be set using the parameter `ISR_E_ORDER` (default: 1). The switch `ISR_E_SCHEME` allows to set the scheme of respecting non-leading terms. Possible options are 0 ("mixed choice"), 1 ("eta choice"), or 2 ("beta choice", default).

**None** No PDF. Fixed beam energy.

Furthermore it is simple to build an external interface to an arbitrary PDF and load that dynamically in the Sherpa run. See *External PDF* for instructions.

**PDF_SET** Specifies the PDF set for hadronic bunch particles. All sets available in the chosen `PDF_LIBRARY` can be figured by running Sherpa with the parameter `SHOW_PDF_SETS: 1`, e.g.:

```
$ Sherpa 'PDF_LIBRARY: CTEQ6Sherpa' 'SHOW_PDF_SETS: 1'
```

If the two colliding beams are of different type, e.g. protons and electrons or photons and electrons, it is possible to specify two different PDF sets by providing two values: `PDF_SET: [pdf1, pdf2]`. The special value `Default` can be used as a placeholder for letting Sherpa choose the appropriate PDF set (or none).

**PDF_SET_VERSIONS** This parameter allows to select a specific version (member) within the chosen PDF set. It is possible to specify two different PDF sets using `PDF_SET_VERSIONS: [version1, version2]`

## 5.4 Models

The main switch `MODEL` sets the model that Sherpa uses throughout the simulation run. The default is `SM`, the built-in Standard Model implementation of Sherpa. For BSM simulations, Sherpa offers an option to use the Universal FeynRules Output Format (UFO) [DDF+12].

Please note: AMEGIC can only be used for the built-in models (SM and HEFT). For anything else, please use Comix.



### 5.4.1 Built-in Models



#### Standard Model

The SM inputs for the electroweak sector can be given in nine different schemes, that correspond to different choices of which SM physics parameters are considered fixed and which are derived from the given quantities. The electro-weak coupling is by default fixed, unless its running has been enabled (cf. *COUPLINGS*). The input schemes are selected through the `EW_SCHEME` parameter, whose default is `alphamZ`. The following options are provided:

**UserDefined** All EW parameters are explicitly given: Here the W, Z and Higgs masses and widths are taken as inputs, and the parameters `1/ALPHAQED(0)`, `ALPHAQED_DEFAULT_SCALE`, `SIN2THETAW` (weak mixing angle), `VEV` (Higgs field vacuum expectation value) and `LAMBDA` (Higgs quartic coupling) have to be specified.

By default, `ALPHAQED_DEFAULT_SCALE: 8315.18` ($= m_Z^2$), which means that the MEs are evaluated with a value of $\alpha = \frac{1}{128.802}$.

Note that this mode allows to violate the tree-level relations between some of the parameters and might thus lead to gauge violations in some regions of phase space.

**alpha0** All EW parameters are calculated from the W, Z and Higgs masses and widths and the fine structure constant (taken from `1/ALPHAQED(0)` + `ALPHAQED_DEFAULT_SCALE`, cf. below) using tree-level relations.

By default, `ALPHAQED_DEFAULT_SCALE: 0.0`, which means that the MEs are evaluated with a value of $\alpha = \frac{1}{137.03599976}$.

**alphamZ** All EW parameters are calculated from the W, Z and Higgs masses and widths and the fine structure constant (taken from `1/ALPHAQED(MZ)`, default `128.802`) using tree-level relations.

**Gmu** This choice corresponds to the G_mu-scheme. The EW parameters are calculated out of the weak gauge boson masses M_W, M_Z, the Higgs boson mass M_H, their respective widths, and the Fermi constant `GF` using tree-level relations.

**alphamZsW** All EW parameters are calculated from the Z and Higgs masses and widths, the fine structure constant (taken from `1/ALPHAQED(MZ)`, default `128.802`), and the weak mixing angle (`SIN2THETAW`) using tree-level relations. In particular, the W boson mass (and in the complex mass scheme also its width) is a derived quantity.

**alphamWsW** All EW parameters are calculated from the W and Higgs masses and widths, the fine structure constant (taken from `1/ALPHAQED(MW)`, default `132.17`), and the weak mixing angle (`SIN2THETAW`) using tree-level relations. In particular, the Z boson mass (and in the complex mass scheme also its width) is a derived quantity.

**GmumZsW** All EW parameters are calculated from the Z and Higgs masses and widths, the Fermi constant (`GF`), and the weak mixing angle (`SIN2THETAW`) using tree-level relations. In particular, the W boson mass (and in the complex mass scheme also its width) is a derived quantity.

**GmumWsW** All EW parameters are calculated from the W and Higgs masses and widths, the Fermi constant (`GF`), and the weak mixing angle (`SIN2THETAW`) using tree-level relations. In particular, the Z boson mass (and in the complex mass scheme also its width) is a derived quantity.

**FeynRules** This choice corresponds to the scheme employed in the FeynRules/UFO setup. The EW parameters are calculated out of the Z boson mass M_Z, the Higgs boson mass M_H, the Fermi constant `GF` and the fine structure constant (taken from `1/ALPHAQED(0)` + `ALPHAQED_DEFAULT_SCALE`, cf. below) using tree-level relations. Note, the W boson mass is not an input parameter in this scheme.

All `Gmu`-derived schemes, where the EW coupling is a derived quantity, possess an ambiguity on how to construct a real EW coupling in the complex mass scheme. Several conventions are implemented and can be accessed through `GMU_CMS_AQED_CONVENTION`.

To account for quark mixing the CKM matrix elements have to be assigned. For this purpose the Wolfenstein parametrization [Wol83] is employed. The order of expansion in the lambda parameter is defined through

```
CKM:
  Order: <order>
  # other CKM settings ...
```

The default for `Order` is `0`, corresponding to a unit matrix. The parameter convention for higher expansion terms reads:

- `Order: 1`, the `Cabibbo` subsetting has to be set, it parametrizes lambda and has the default value `0.22537`.
- `Order: 2`, in addition the value of `CKM_A` has to be set, its default is `0.814`.
- `Order: 3`, the order lambda^3 expansion, `Eta` and `Rho` have to be specified. Their default values are `0.353` and `0.117`, respectively.

The CKM matrix elements V_ij can also be read in using

```
CKM:
  Matrix_Elements:
    i,j: <V_ij>
    # other CKM matrix elements ...
  # other CKM settings ...
```

Complex values can be given by providing two values: `<V_ij> -> [Re, Im]`. Values not explicitly given are taken from the afore computed Wolfenstein parametrisation. Setting `CKM: {Output: true}` enables an output of the CKM matrix.

The remaining parameter to fully specify the Standard Model is the strong coupling constant at the Z-pole, given through `ALPHAS(MZ)`. Its default value is `0.118`. If the setup at hand involves hadron collisions and thus PDFs, the value of the strong coupling constant is automatically set consistent with the PDF fit and can not be changed by the user. If Sherpa is compiled with LHAPDF support, it is also possible to use the alphaS evolution provided in LHAPDF by specifying `ALPHAS: {USE_PDF: 1}`. The perturbative order of the running of the strong coupling can be set via `ORDER_ALPHAS`, where the default `0` corresponds to one-loop running and `1`, `2`, `3` to `2,3,4`-loops, respectively. If the setup at hand involves PDFs, this parameter is set consistent with the information provided by the PDF set.

If unstable particles (e.g. W/Z bosons) appear as intermediate propagators in the process, Sherpa uses the complex mass scheme to construct MEs in a gauge-invariant way. For full consistency with this scheme, by default the dependent EW parameters are also calculated from the complex masses (`WIDTH_SCHEME: CMS`), yielding complex values e.g. for the weak mixing angle. To keep the parameters real one can set `WIDTH_SCHEME: Fixed`. This may spoil gauge invariance though.

With the following switches it is possible to change the properties of all fundamental particles:

```
PARTICLE_DATA:
  <id>:
    <Property>: <value>
    # other properties for this particle ...
  # data for other particles
```

Here, `<id>` is the PDG ID of the particle for which one more properties are to be modified. `<Property>` can be one of the following:

**`Mass`** Sets the mass (in GeV) of the particle.

Masses of particles and corresponding anti-particles are always set simultaneously.

For particles with Yukawa couplings, those are enabled/disabled consistent with the mass (taking into account the `Massive` parameter) by default, but that can be modified using the `Yukawa` parameter. Note that by default the Yukawa couplings are treated as running, cf. *YUKAWA_MASSES*.

**`Massive`** Specifies whether the finite mass of the particle is to be considered in matrix-element calculations or not. Can be `true` or `false`.

**`Width`** Sets the width (in GeV) of the particle.

**`Active`** Enables/disables the particle with PDG id `<id>`. Can be `true` or `false`.

**`Stable`** Sets the particle either stable or unstable according to the following options:

- **`0`** Particle and anti-particle are unstable
- **`1`** Particle and anti-particle are stable
- **`2`** Particle is stable, anti-particle is unstable
- **`3`** Particle is unstable, anti-particle is stable

This option applies to decays of hadrons (cf. *Hadron decays*) as well as particles produced in the hard scattering (cf. *Hard decays*). For the latter, alternatively the decays can be specified explicitly in the process setup (see *Processes*) to avoid the narrow-width approximation.

**`Priority`** Allows to overwrite the default automatic flavour sorting in a process by specifying a priority for the given flavour. This way one can identify certain particles which are part of a container (e.g. massless b-quarks), such that their position can be used reliably in selectors and scale setters.

---

**Note:** `PARTICLE_DATA` can also be used to the properties of hadrons, you can use the same switches (except for `Massive`), see *Hadronization*.

---

#### Effective Higgs Couplings

The HEFT describes the effective coupling of gluons and photons to Higgs bosons via a top-quark loop, and a W-boson loop in case of photons. This supplement to the Standard Model can be invoked by configuring `MODEL: HEFT`.

The effective coupling of gluons to the Higgs boson, g_ggH, can be calculated either for a finite top-quark mass or in the limit of an infinitely heavy top using the switch `FINITE_TOP_MASS: true` or `FINITE_TOP_MASS: false`, respectively. Similarily, the photon-photon-Higgs coupling, g_ppH, can be calculated both for finite top and/or W masses or in the infinite mass limit using the switches `FINITE_TOP_MASS` and `FINITE_W_MASS`. The default choice for both is the infinite mass limit in either case. Note that these switches affect only the calculation of the value of the effective coupling constants. Please refer to the example setup *H+jets production in gluon fusion with finite top mass effects* for information on how to include finite top quark mass effects on a differential level.

Either one of these couplings can be switched off using the `DEACTIVATE_GGH: true` and `DEACTIVATE_PPH: true` switches. Both default to `false`.

### 5.4.2 UFO Model Interface

To use a model generated by the FeynRules package [CD09],:cite:*Christensen2009jx*, the model must be made available to Sherpa by running

```
$ <prefix>/bin/Sherpa-generate-model <path-to-ufo-model>
```

where `<path-to-ufo-model>` specifies the location of the directory where the UFO model can be found. UFO support must be enabled using the `--enable-ufo` option of the configure script, as described in *Installation*. This requires Python version 2.6 or later and an installation of SCons.

The above command generates source code for the UFO model, compiles it, and installs the corresponding library, making it available for event generation. Python, SCons, and the UFO model directory are not required for event generation once the above command has finished. Note that the installation directory for the created library and the paths to Sherpa libraries and headers are predetermined automatically durin the installation of Sherpa. If the Sherpa installation is moved afterwards or if the user does not have the necessary permissions to install the new library in the predetermined location, these paths can be set manually.

Please run

```
$ <prefix>/bin/Sherpa-generate-model --help
```

for information on the relevan command line arguments.

An example configuration file will be written to the working directory while the model is generated with `Sherpa-generate-model`. This config file shows the syntax for the respective model parameters and can be used as a template. It is also possible to use an external parameter file by specifying the path to the file with the switch

`UFO_PARAM_CARD` in the configuration file or on the command line. Relative and absolute file paths are allowed. This option allows it to use the native UFO parameter cards, as used by MadGraph for example.

Note that the use of the SM `PARTICLE_DATA` switches `Mass`, `Massive`, `Width`, and `Stable` is discouraged when using UFO models as the UFO model completely defines all particle properties and their relation to the independent model parameters. These model parameters should be set using the standard UFO parameter syntax as shown in the example run card generated by the `Sherpa-generate-model` command.

For parts of the simulation other than the hard process (hadronization, underlying event, running of the SM couplings) Sherpa uses internal default values for the Standard Model fermion masses if they are massless in the UFO model. This is necessary for a meaningful simulation. In the hard process however, the UFO model masses are always respected.

For an example UFO setup, see *Event generation in the MSSM using UFO*. For more details on the Sherpa interface to FeynRules please consult [CdAD+11],:cite:*Hoeche2014kca*.

Please note that AMEGIC can only be used for the built-in models (SM and HEFT). The use of UFO models is only supported by Comix.

## 5.5 Matrix elements

The following parameters are used to steer the matrix element calculation setup. To learn how to specify the hard scattering process and further process-specific options in its calculation, please also refer to *Processes*.

### 5.5.1 ME_GENERATORS

The list of matrix element generators to be employed during the run. When setting up hard processes, Sherpa calls these generators in order to check whether either one is capable of generating the corresponding matrix element. This parameter can also be set on the command line using option `-m`, see *Command Line Options*.

The built-in generators are

**`Internal`** Simple matrix element library, implementing a variety of 2->2 processes.

**`Amegic`** The AMEGIC++ generator published under [KKS02]

**`Comix`** The Comix generator published under [GH08]

It is possible to employ an external matrix element generator within Sherpa. For advice on this topic please contact the authors, *Authors*.

### 5.5.2 RESULT_DIRECTORY

This parameter specifies the name of the directory which is used by Sherpa to store integration results and phasespace mappings. The default is `Results/`. It can also be set using the command line parameter `-r`, see *Command Line Options*. The directory will be created automatically, unless the option `GENERATE_RESULT_DIRECTORY: false` is specified. Its location is relative to a potentially specified input path, see *Command Line Options*.

### 5.5.3 EVENT_GENERATION_MODE

This parameter specifies the event generation mode. It can also be set on the command line using option `-w`, see *Command Line Options*. The three possible options are:

**`Weighted`** (alias `W`) Weighted events.

**`Unweighted`** (alias `U`) Events with constant weight, which have been unweighted against the maximum determined during phase space integration. In case of rare events with `w > max` the parton level event is repeated `floor(w/max)` times and the remainder is unweighted. While this leads to unity weights for all events it can be misleading since the statistical impact of a high-weight event is not accounted for. In the extreme case this can lead to a high-weight event looking like a significant bump in distributions (in particular after the effects of the parton shower).

**`PartiallyUnweighted`** (alias `P`) Identical to `Unweighted` events, but if the weight exceeds the maximum determined during the phase space integration, the event will carry a weight of `w/max` to correct for that. This is the recommended option to generate unweighted events and the default setting in Sherpa.

For `Unweighted` and `PartiallyUnweighted` events the user may set `OVERWEIGHT_THRESHOLD:` to cap the maximal over-weight `w/max` taken into account.

### 5.5.4 SCALES

This parameter specifies how to compute the renormalization and factorization scale and potential additional scales.

---

**Note:** In a setup with the parton shower enabled, it is strongly recommended to leave this at its default value, `METS`, and to instead customise the `CORE_SCALE` setting as described in *METS scale setting with multiparton core processes*.

---

Sherpa provides several built-in scale setting schemes. For each scheme the scales are then set using expressions understood by the *Interpreter*. Each scale setter's syntax is

```
SCALES: <scale-setter>{<scale-definition>}
```

to define a single scale for both the factorisation and renormalisation scale. They can be set to different values using

```
SCALES: <scale-setter>{<fac-scale-definition>}{<ren-scale-definition>}
```

In parton shower matched/merged calculations a third perturbative scale is present, the resummation or parton shower starting scale. It can be set by the user in the third argument like

```
SCALES: <scale-setter>{<fac-scale-definition>}{<ren-scale-definition>}{<res-scale-
↪definition>}
```

If the final state of your hard scattering process contains QCD partons, their kinematics fix the resummation scale for subsequent emissions (cf. the description of the `METS` scale setter below). With the CS Shower, you can instead specify your own resummation scale also in such a case: Set `CSS_RESPECT_Q2: true` and use the third argument to specify your resummation scale as above.

**Note:** For all scales their squares have to be given. See *Predefined scale tags* for some predefined scale tags.

More than three scales can be set as well to be subsequently used, e.g. by different couplings, see *COUPLINGS*.

### Scale setters

The scale setter options which are currently available are

**VAR** The variable scale setter is the simplest scale setter available. Scales are simply specified by additional parameters in a form which is understood by the internal interpreter, see *Interpreter*. If, for example the invariant mass of the lepton pair in Drell-Yan production is the desired scale, the corresponding setup reads

```
SCALES: VAR{Abs2(p[2]+p[3])}
```

Renormalization and factorization scales can be chosen differently. For example in Drell-Yan + jet production one could set

```
SCALES: VAR{Abs2(p[2]+p[3])}{MPerp2(p[2]+p[3])}
```

**FASTJET** If FastJet is enabled by including `--enable-fastjet=/path/to/fastjet` in the `configure` options, this scale setter can be used to set a scale based on jet-, rather than parton-momenta.

The final state parton configuration is first clustered using FastJet and resulting jet momenta are then added back to the list of non strongly interacting particles. The numbering of momenta therefore stays effectively the same

as in standard Sherpa, except that final state partons are replaced with jets, if applicable (a parton might not pass the jet criteria and get "lost"). In particular, the indices of the initial state partons and all EW particles are uneffected. Jet momenta can then be accessed as described in *Predefined scale tags* through the identifiers `p[i]`, and the nodal values of the clustering sequence can be used through `MU_n2`. The syntax is

```
SCALES: FASTJET[<jet-algo-parameter>]{<scale-definition>}
```

Therein the parameters of the jet algorithm to be used to define the jets are given as a comma separated list of

- the jet algorithm `A:kt,antikt,cambridge,siscone` (default `antikt`)
- phase space restrictions, i.e. `PT:<min-pt>`, `ET:<min-et>`, `Eta:<max-eta>`, `Y:<max-rap>` (otherwise unrestricted)
- radial parameter `R:<rad-param>` (default `0.4`)
- f-parameter for Siscone `f:<f-param>` (default `0.75`)
- recombination scheme `C:E,pt,pt2,Et,Et2,BIpt,BIpt2` (default `E`)
- b-tagging mode `B:0,1,2` (default `0`) This parameter, if specified different from its default 0, allows to use b-tagged jets only, based on the parton-level constituents of the jets. There are two options: With `B:1` both b and anti-b quarks are counted equally towards b-jets, while for `B:2` they are added with a relative sign as constituents, i.e. a jet containing b and anti-b is not tagged.
- scale setting mode `M:0,1` (default `1`) It is possible to specify multiple scale definition blocks, each enclosed in curly brackets. The scale setting mode parameter then determines, how those are interpreted: In the `M:0` case, they specify factorisation, renormalisation and resummation scale separately in that order. In the `M:1` case, the `n` given scales are used to calculate a mean scale such that $\alpha_s^n(\mu_{\text{mean}}) = \alpha_s(\mu_1)\dots\alpha_s(\mu_n)$ This scale is then used for factorisation, renormalisation and resummation scale.

Consider the example of lepton pair production in association with jets. The following scale setter

```
SCALES: FASTJET[A:kt,PT:10,R:0.4,M:0]{sqrt(PPerp2(p[4])*PPerp2(p[5]))}
```

reconstructs jets using the kt-algorithm with R=0.4 and a minimum transverse momentum of 10 GeV. The scale of all strong couplings is then set to the geometric mean of the hardest and second hardest jet. Note `M:0`.

Similarly, in processes with multiple strong couplings, their renormalisation scales can be set to different values, e.g.

```
SCALES: FASTJET[A:kt,PT:10,R:0.4,M:1]{PPerp2(p[4])}{PPerp2(p[5])}
```

sets the scale of one strong coupling to the transverse momentum of the hardest jet, and the scale of the second strong coupling to the transverse momentum of second hardest jet. Note `M:1` in this case.

The additional tags *MU_22* .. *MU_n2* (n=2..njet+1), hold the nodal values of the jet clustering in descending order.

Please note that currently this type of scale setting can only be done within the process block (*Processes*) and not within the (me) section.

**METS** The matrix element is clustered onto a core 2->2 configuration using an inversion of current parton shower, cf. *SHOWER_GENERATOR*, recombining (n+1) particles into n on-shell particles. Their corresponding flavours are determined using run-time information from the matrix element generator. It defines the three tags `MU_F2`, `MU_R2` and `MU_Q2` whose values are assigned through this clustering procedure. While `MU_F2` and `MU_Q2` are defined as the lowest invariant mass or negative virtuality in the core process (for core interactions which are pure QCD processes scales are set to the maximum transverse mass squared of the outgoing particles), `MU_R2` is determined using this core scale and the individual clustering scales such that

$$\alpha_s(\mu_{R2})^{n+k} = \alpha_s(\text{core} - \text{scale})^k \alpha_s(kt_1)\dots\alpha_s(kt_n)$$

where k is the order in strong coupling of the core process and k is the number of clusterings, kt_i are the relative transverse momenta at each clustering. The tags `MU_F2`, `MU_R2` and `MU_Q2` can then be used on equal footing with the tags of *Predefined scale tags* to define the final scale.

`METS` is the default scale scheme in Sherpa, since it is employed for truncated shower merging, see *Multijet merged event generation with Sherpa*, both at leading and next-to-leading order. Thus, Sherpa's default is

```
SCALES: METS{MU_F2}{MU_R2}{MU_Q2}
```

As the tags `MU_F2`, `MU_R2` and `MU_Q2` are predefined by the `METS` scale setter, they may be omitted, i.e.

```
SCALES: METS
```

leads to an identical scale definition.

The `METS` scale setter comes in two variants: `STRICT_METS` and `LOOSE_METS`. While the former employs the exact inverse of the parton shower for the clustering procedure, and therefore is rather time consuming for multiparton final state, the latter is a simplified version and much faster. Giving `METS` as the scale setter results in using `LOOSE_METS` for the integration and `STRICT_METS` during event generation. Giving either `STRICT_METS` or `LOOSE_METS` as the scale setter results in using the respective one during both integration and event generation.

Clusterings onto 2->n (n>2) configurations is possible, see *METS scale setting with multiparton core processes*.

This scheme might be subject to changes to enable further classes of processes for merging in the future and should therefore be seen with care. Integration results might change slightly between different Sherpa versions.

Occasionally, users might encounter the warning message

```
METS_Scale_Setter::CalculateScale(): No CSS history for '<process name>' in
↪<percentage>% of calls. Set \hat{s}.
```

As long as the percentage quoted here is not too high, this does not pose a serious problem. The warning occurs when - based on the current colour configuration and matrix element information - no suitable clustering is found by the algorithm. In such cases the scale is set to the invariant mass of the partonic process.

### Custom scale implementation

When the flexibility of the `VAR` scale setter above is not sufficient, it is also possible to implement a completely custom scale scheme within Sherpa as C++ class plugin. For details please refer to the *Customization* section.

### Predefined scale tags

There exist a few predefined tags to facilitate commonly used scale choices or easily implement a user defined scale.

**`p[n]`** Access to the four momentum of the nth particle. The initial state particles carry n=0 and n=1, the final state momenta start from n=2. Their ordering is determined by Sherpa's internal particle ordering and can be read e.g. from the process names displayed at run time. Please note, that when building jets out of the final state partons first, e.g. through the `FASTJET` scale setter, these parton momenta will be replaced by the jet momenta ordered in transverse momenta. For example the process u ub -> e- e+ G G will have the electron and the positron at positions `p[2]` and `p[3]` and the gluons on postions `p[4]` and `p[5]`. However, when finding jets first, the electrons will still be at `p[2]` and `p[3]` while the harder jet will be at `p[4]` and the softer one at `p[5]`.

**`H_T2`** Square of the scalar sum of the transverse momenta of all final state particles.

**`H_TM2`** Square of the scalar sum of the transverse energies of all final state particles, i.e. contrary to `H_T2` `H_TM2` takes particle masses into account.

**H_TY2(<factor>,<exponent>)** Square of the scalar sum of the transverse momenta of all final state particles weighted by their rapidity distance from the final state boost vector. Thus, takes the form

```
H_T^{(Y)} = sum_i pT_i exp [ fac |y-yboost|^exp ]
```

Typical values to use would by `0.3` and `1`.

**H_Tp2** Scale setter for lepton-pair production in association with jets only, implements

```
H_T' = sqrt(m_ll^2 + pT(ll)^2) + sum_i pT_i (i not l)
```

**DH_Tp2(<recombination-method>,<dR>)** Implements a version of `H_Tp2` which dresses charged particles first. The parameter `<recombination-method>` can take the following values: `Cone`, `kt`, `CA` or `antikt`, while `<dR>` is the respecitve algorithm's angular distance parameter.

**TAU_B2** Square of the beam thrust.

**MU_F2, MU_R2, MU_Q2** Tags holding the values of the factorisation, renormalisation scale and resummation scale determined through backwards clustering in the `METS` scale setter.

**MU_22, MU_32, ..., MU_n2** Tags holding the nodal values of the jet clustering in the `FASTJET` scale setter, cf. *Scale setters*.

All of those objects can be operated upon by any operator/function known to the *Interpreter*.

### Scale schemes for NLO calculations

For next-to-leading order calculations it must be guaranteed that the scale is calculated separately for the real correction and the subtraction terms, such that within the subtraction procedure the same amount is subtracted and added back. Starting from version 1.2.2 this is the case for all scale setters in Sherpa. Also, the definition of the scale must be infrared safe w.r.t. to the radiation of an extra parton. Infrared safe (for QCD-NLO calculations) are:

- any function of momenta of NOT strongly interacting particles
- sum of transverse quantities of all partons (e.g. `H_T2`)
- any quantity refering to jets, constructed by an IR safe jet algorithm, see below.

Not infrared safe are

- any function of momenta of specific partons
- for processes with hadrons in the initial state:

any quantity that depends on parton momenta along the beam axis, including the initial state partons itself

Since the total number of partons is different for different pieces of the NLO calculation any explicit reference to a parton momentum will lead to an inconsistent result.

### Explicit scale variations

The factorisation and renormalisation scales in the fixed-order matrix elements can be varied separately simply by introducing a prefactor into the scale definition, e.g.

```
SCALES: VAR{0.25*H_T2}{0.25*H_T2}
```

for setting both the renormalisation and factorisation scales to H_T/2.

Similarly, the starting scale of the parton shower resummation in a ME+PS merged sample can be varied using the METS scale setter's third argument like:

```
SCALES: METS{MU_F2}{MU_R2}{4.0*MU_Q2}
```

#### METS scale setting with multiparton core processes

The METS scale setter stops clustering when no combination is found that corresponds to a parton shower branching, or if two subsequent branchings are unordered in terms of the parton shower evolution parameter. The core scale of the remaining 2->n process then needs to be defined. This is done by specifying a core scale through

```
CORE_SCALE: <core-scale-setter>{<core-fac-scale-definition>}{<core-ren-scale-
↪definition>}{<core-res-scale-definition>}
```

As always, for scale setters which define `MU_F2`, `MU_R2` and `MU_Q2` the scale definition can be dropped. Possible core scale setters are

**VAR** Variable core scale setter. Syntax is identical to variable scale setter.

**QCD** QCD core scale setter. Scales are set to harmonic mean of s, t and u. Only useful for 2->2 cores as alternatives to the usual core scale of the METS scale setter.

**TTBar** Core scale setter for processes involving top quarks. Implementation details are described in Appendix C of [HHL+13].

**SingleTop** Core scale setter for single-top production in association with one jet. If the W is in the t-channel (s-channel), the squared scales are set to the Mandelstam variables `t=2*p[0]*p[2]` (`t=2*p[0]*p[1]`).

### 5.5.5 COUPLINGS

Within Sherpa, strong and electroweak couplings can be computed at any scale specified by a scale setter (cf. *SCALES*). The `COUPLINGS` tag links the argument of a running coupling to one of the respective scales. This is better seen in an example. Assuming the following input

```
SCALES: VAR{...}{PPerp2(p[2])}{Abs2(p[2]+p[3])}
COUPLINGS:
  - "Alpha_QCD 1"
  - "Alpha_QED 2"
```

Sherpa will compute any strong couplings at scale one, i.e. `PPerp2(p[2])` and electroweak couplings at scale two, i.e. `Abs2(p[2]+p[3])`. Note that counting starts at zero.

### 5.5.6 KFACTOR

This parameter specifies how to evaluate potential K-factors in the hard process. This is equivalent to the `COUPLINGS` specification of Sherpa versions prior to 1.2.2. Currently available options are

**None** No reweighting

**VAR** Couplings specified by an additional parameter in a form which is understood by the internal interpreter, see *Interpreter*. The tags `Alpha_QCD` and `Alpha_QED` serve as links to the built-in running coupling implementation.

If for example the process `g g -> h g` in effective theory is computed, one could think of evaluating two powers of the strong coupling at the Higgs mass scale and one power at the transverse momentum squared of the gluon. Assuming the Higgs mass to be 120 GeV, the corresponding reweighting would read

```
SCALES:     VAR{...}{PPerp2(p[3])}
COUPLINGS: "Alpha_QCD 1"
KFACTOR:    VAR{sqr(Alpha_QCD(sqr(120))/Alpha_QCD(MU_12))}
```

As can be seen from this example, scales are referred to as `MU_<i>2`, where `<i>` is replaced with the appropriate number. Note that counting starts at zero.

It is possible to implement a dedicated K-factor scheme within Sherpa. For advice on this topic please contact the authors, *Authors*.

### 5.5.7 YUKAWA_MASSES

This parameter specifies whether the Yukawa couplings are evaluated using running or fixed quark masses: `YUKAWA_MASSES: Running` is the default since version 1.2.2 while `YUKAWA_MASSES: Fixed` was the default until 1.2.1.

### 5.5.8 Dipole subtraction

This list of parameters can be used to optimize the performance when employing the Catani-Seymour dipole subtraction [CS97] as implemented in Amegic [GK08]. The dipole parameters are specified as subsettings to the `DIPOLES` setting, like this:

```
DIPOLES:
  ALPHA: <alpha>
  NF_GSPLIT: <nf>
  # other dipole settings ...
```

The following parameters can be customised:

**LPHA** Specifies a dipole cutoff in the nonsingular region [Nag03]. Changing this parameter shifts contributions from the subtracted real correction piece (RS) to the piece including integrated dipole terms (I), while their sum remains constant. This parameter can be used to optimize the integration performance of the individual pieces. Also the average calculation time for the subtracted real correction is reduced with smaller choices of "ALPHA" due to the (on average) reduced number of contributing dipole terms. For most processes a reasonable choice is between 0.01 and 1 (default). See also *Choosing DIPOLES ALPHA*

**AMIN** Specifies the cutoff of real correction terms in the infrared reagion to avoid numerical problems with the subtraction. The default is 1.e-8.

**NF_GSPLIT** Specifies the number of quark flavours that are produced from gluon splittings. This number must be at least the number of massless flavours (default). If this number is larger than the number of massless quarks the massive dipole subtraction [CDST02] is employed.

**KAPPA** Specifies the kappa-parameter in the massive dipole subtraction formalism [CDST02]. The default is 2.0/3.0.

## 5.6 Processes

In addition to the general matrix-element calculation settings specified as described in *Matrix elements*, the hard scattering process has to be defined and further process-specific calculation settings can be specified. This happens in the `PROCESSES` part of the input file and is described in the following section.

A simple example looks like:

```
PROCESSES:
- 93 93 -> 11 -11 93{4}:
    Order: {QCD: 0, EW: 2}
    CKKW: 20
```

In general, the process setup takes the following form:

```
PROCESSES:
# Process 1:
- <process declaration>:
    <parameter>: <value>
    <multiplicities-to-be-applied-to>:
      <parameter>: <value>
      ...
# Process 2:
- <process declaration>
    ...
```

i.e. `PROCESSES` followed by a list of process definitions.

Each process definition starts with the declaration of the (core) process itself. The initial and final state particles are specified by their PDG codes, or by particle containers, see *Particle containers*. Examples are

**`- 93 93 -> 11 -11`** Sets up a Drell-Yan process group with light quarks in the initial state.

**`- 11 -11 -> 93 93 93{3}`** Sets up jet production in e+e- collisions with up to three additional jets.

Special features of the process declaration will be documented in the following. The remainder of the section then documents all additional parameters for the process steering, e.g. the coupling order, which can be nested as `key: value` pairs within a given process declaration.

An advanced syntax feature shall be mentioned already here, since it will be used in some of the examples in the following: Most of the parameters can be grouped under a multiplicity key, which can either be a single or a range of multiplicities, e.g. `2->2-4: { <settings for 2->2, 2->3 and 2->4 processes> }`. The usefulness of this will hopefully become clear in the examples in the following.

- *Order*
- *Max_Order*
- *Min_Order*
- *Min_N_Quarks*
- *Max_N_Quarks*
- *Min_N_TChannels*
- *Max_N_TChannels*
- *Print_Graphs*
- *Name_Suffix*
- *Integration_Error*
- *Max_Epsilon*
- *NLO_Mode*
- *NLO_Part*
- *NLO_Order*
- *Subdivide_Virtual*
- *ME_Generator*
- *RS_ME_Generator*
- *Loop_Generator*
- *Integrator*
- *PSI_ItMin*
- *RS_PSI_ItMin*
- *Special Group*
- *Event biasing*

### 5.6.1 Special features of the process declaration

- *PDG codes*
- *Particle containers*
- *Parentheses*
- *Curly brackets*

### PDG codes

Initial and final state particles are specified using their PDG codes (cf. PDG). A list of particles with their codes, and some of their properties, is printed at the start of each Sherpa run, when the *OUTPUT* is set at level 2.

### Particle containers

Sherpa contains a set of containers that collect particles with similar properties, namely

- lepton (carrying number 90),
- neutrino (carrying number 91),
- fermion (carrying number 92),
- jet (carrying number 93),
- quark (carrying number 94).

These containers hold all massless particles and anti-particles of the denoted type and allow for a more efficient definition of initial and final states to be considered. The jet container consists of the gluon and all massless quarks, as set by

```
PARTICLE_DATA:
  <id>:
    Mass: 0
    # ... and/or ...
    Massive: false
```

A list of particle containers is printed at the start of each Sherpa run, when the *OUTPUT* is set at level 2.

It is also possible to define a custom particle container using the keyword PARTICLE_CONTAINER. The container must be given an unassigned particle ID (kf-code) and its name (freely chosen by you) and the flavour content must be specified. An example would be the collection of all down-type quarks using the unassigned ID 98, which could be declared as

```
PARTICLE_CONTAINER:
  98:
    Name: downs
    Flavours: [1, -1, 3, -3, 5, -5]
```

Note that, if wanted, you have to add both particles and anti-particles.

### Parentheses

The parenthesis notation allows to group a list of processes with different flavor content but similar structure. This is most useful in the context of simulations containing heavy quarks. In a setup with massive b-quarks, for example, the b-quark will not be part of the jets container. In order to include b-associated processes easily, the following can be used:

```
PARTICLE_DATA:
  5: {Massive: true}
PARTICLE_CONTAINER:
  98: {Name: B, Flavours: [5, -5]}
PROCESSES:
- 11 -11 -> (93,98) (93,98):
  ...
```

#### Curly brackets

The curly bracket notation when specifying a process allows up to a certain number of jets to be included in the final state. This is easily seen from an example, `11 -11 -> 93 93 93{3}` sets up jet production in e+e- collisions. The matix element final state may be 2, 3, 4 or 5 light partons or gluons.

### 5.6.2 Decay

Specifies the exclusive decay of a particle produced in the matrix element. The virtuality of the decaying particle is sampled according to a Breit-Wigner distribution. In practice this amounts to selecting only those diagrams containing s-channels of the specified flavour while the phase space is kept general. Consequently, all spin correlations are preserved. An example would be

```
- 11 -11 -> 6[a] -6[b]:
   Decay:
   - 6[a] -> 5 24[c]
   - -6[b] -> -5 -24[d]
   - 24[c] -> -13 14
   - -24[d] -> 94 94
```

### 5.6.3 DecayOS

Specifies the exclusive decay of a particle produced in the matrix element. The decaying particle is on mass-shell, i.e. a strict narrow-width approximation is used. This tag can be specified alternatively as `DecayOS`. In practice this amounts to selecting only those diagrams containing s-channels of the specified flavour and the phase space is factorised as well. Nonetheless, all spin correlations are preserved. An example would be

```
- 11 -11 -> 6[a] -6[b]:
    DecayOS:
    - 6[a] -> 5 24[c]
    - -6[b] -> -5 -24[d]
    - 24[c] -> -13 14
    - -24[d] -> 94 94
```

### 5.6.4 No_Decay

Remove all diagrams associated with the decay/s-channel of the given flavours. Serves to avoid resonant contributions in processes like W-associated single-top production. Note that this method breaks gauge invariance! At the moment this flag can only be set for Comix. An example would be

```
- 93 93 -> 6[a] -24[b] 93{1}:
    Decay: 6[a] -> 5 24[c]
    DecayOS:
    - 24[c] -> -13 14
    - -24[b] -> 11 -12
    No_Decay: -6
```

### 5.6.5 Scales

Sets a process-specific scale. For the corresponding syntax see *SCALES*.

### 5.6.6 Couplings

Sets process-specific couplings. For the corresponding syntax see *COUPLINGS*.

### 5.6.7 CKKW

Sets up multijet merging according to [HKSS09]. The additional argument specifies the parton separation criterion ("merging cut") Q_{cut} in GeV. It can be given in any form which is understood by the internal interpreter, see *Interpreter*. Examples are

- Hadronic collider: `CKKW: 20`
- Leptonic collider: `CKKW: pow(10,-2.5/2.0)*E_CMS`
- DIS: `CKKW: $(QCUT)/sqrt(1.0+sqr($(QCUT)/$(SDIS))/Abs2(p[2]-p[0]))`

### 5.6.8 Process_Selectors

Using `Selectors: [<selector 1>, <selector 2>]` in a process definition sets up process-specific selectors. They use the same syntax as describes in *Selectors*.

### 5.6.9 Order

Sets a process-specific coupling order. Orders are counted at the amplitude level. For example, the process 1 -1 -> 2 -2 would have orders `{QCD: 2, EW: 0}`, `{QCD: 1, EW: 1}` and `{QCD: 0, EW: 2}`. There can also be a third entry that is model specific (e.g. for HEFT couplings). Half-integer orders are so far supported only by Comix. The word "Any" can be used as a wildcard.

Note that for decay chains this setting applies to the full process, see *Decay* and *DecayOS*.

### 5.6.10 Max_Order

Sets a process-specific maximum coupling order. See *Order* for the syntax and additional information.

### 5.6.11 Min_Order

Sets a process-specific minimum coupling order. See *Order* for the syntax and additional information.

### 5.6.12 Min_N_Quarks

Limits the minimum number of quarks in the process to the given value.

### 5.6.13 Max_N_Quarks

Limits the maximum number of quarks in the process to the given value.

### 5.6.14 Min_N_TChannels

Limits the minimum number of t-channel propagators in the process to the given value.

### 5.6.15 Max_N_TChannels

Limits the maximum number of t-channel propagators in the process to the given value.

### 5.6.16 Print_Graphs

Writes out Feynman graphs in LaTeX format. The parameter specifies a directory name in which the diagram information is stored. This directory is created automatically by Sherpa. The LaTeX source files can be compiled using the command

```
$ ./plot_graphs <graphs directory>
```

which creates an html page in the graphs directory that can be viewed in a web browser.

### 5.6.17 Name_Suffix

Defines a unique name suffix for the process.

### 5.6.18 Integration_Error

Sets a process-specific relative integration error target. An example to specify an error target of 2% for 2->3 and 2->4 processes would be:

```
- 93 93 -> 93 93 93{2}:
    2->3-4:
      Integration_Error: 0.02
```

### 5.6.19 Max_Epsilon

Sets epsilon for maximum weight reduction. The key idea is to allow weights larger than the maximum during event generation, as long as the fraction of the cross section represented by corresponding events is at most the epsilon factor times the total cross section. In other words, the relative contribution of overweighted events to the inclusive cross section is at most epsilon.

### 5.6.20 NLO_Mode

This setting specifies whether and in which mode an NLO calculation should be performed. Possible values are:

**None** perform a leading-order calculation (this is the default)

**Fixed_Order** perform a fixed-order next-to-leading order calculation

**MC@NLO** perform an MC@NLO-type matching of a fixed-order next-to-leading order calculation to the resummation of the parton shower

The usual multiplicity identifier applies to this switch as well. Note that using a value other than `None` implies `NLO_Part: BVIRS` for the relevant multiplicities. For fixed-order NLO calculations (`NLO_Mode: Fixed_Order`), this can be overridden by setting `NLO_Part` explicitly, see *NLO_Part*.

Note that Sherpa includes only a very limited selection of one-loop corrections. For processes not included external codes can be interfaced, see *External one-loop ME*

### 5.6.21 NLO_Part

In case of fixed-order NLO calculations this switch specifies which pieces of a NLO calculation are computed, also see *NLO_Mode*. Possible choices are

**B** born term

**V** virtual (one-loop) correction

**I** integrated subtraction terms

**RS** real correction, regularized using Catani-Seymour subtraction terms

Different pieces can be combined in one processes setup. Only pieces with the same number of final state particles and the same order in alpha_S and alpha can be treated as one process, otherwise they will be automatically split up.

### 5.6.22 NLO_Order

Specifies the relative order of the NLO correction wrt. the considered Born process. For example, `NLO_Order: {QCD: 1, EW: 0}` specifies a QCD correction while `NLO_Order: {QCD: 0, EW: 1}` specifies an EW correction.

### 5.6.23 Subdivide_Virtual

Allows to split the virtual contribution to the total cross section into pieces. Currently supported options when run with BlackHat are `LeadingColor` and `FullMinusLeadingColor`. For high-multiplicity calculations these settings allow to adjust the relative number of points in the sampling to reduce the overall computation time.

### 5.6.24 ME_Generator

Set a process specific nametag for the desired tree-ME generator, see *ME_GENERATORS*.

### 5.6.25 RS_ME_Generator

Set a process specific nametag for the desired ME generator used for the real minus subtraction part of NLO calculations. See also *ME_GENERATORS*.

### 5.6.26 Loop_Generator

Set a process specific nametag for the desired loop-ME generator. The only Sherpa-native option is `Internal` with a few hard coded loop matrix elements. Other loop matrix elements are provided by external libraries.

### 5.6.27 Integrator

Sets a process-specific integrator, see *INTEGRATOR*.

### 5.6.28 PSI_ItMin

Sets the number of points per optimization step, see *PSI*.

### 5.6.29 RS_PSI_ItMin

Sets the number of points per optimization step in real-minus-subtraction parts of fixed-order and MC@NLO calculations, see *PSI*.

### 5.6.30 Special Group

**Note:** Needs update to Sherpa 3.x YAML syntax.

Allows to split up individual flavour processes within a process group for integrating them separately. This can help improve the integration/unweighting efficiency. Note: Only works with Comix so far. Example for usage:

```
Process 93 93 -> 11 -11 93
Special Group(0-1,4)
[...]
End process
Process 93 93 -> 11 -11 93
Special Group(2-3,5-7)
```

(continues on next page)

(continued from previous page)

```
[...]
End process
```

The numbers for each individual process can be found using a script in the AddOns directory: `AddOns/ShowProcessIds.sh Process/Comix.zip`

### 5.6.31 Event biasing

In the default event generation mode, events will be distributed "naturally" in the phase space according to their differential cross sections. But sometimes it is useful, to statistically enhance the event generation for rare phase space regions or processes/multiplicities. This is possible with the following options in Sherpa. The generation of more events in a rare region will then be compensated through event weights to yield the correct differential cross section. These options can be applied both in weighted and unweighted event generation.



#### Enhance_Factor

Factor with which the given *process/multiplicity* should be statistically biased. In the following example, the Z+1j process is generated 10 times more often than naturally, compared to the Z+0j process. Each Z+1j event will thus receive a weight of 1/10 to compensate for the bias.

```
- 93 93 -> 11 -11 93{1}:
    2->3:
      Enhance_Factor: 10.0
```

#### RS_Enhance_Factor

Sets an enhance factor (see *Enhance_Factor*) for the RS-piece of an MC@NLO process.

#### Enhance_Function

Specifies a phase-space dependent biasing of parton-level events (before showering). The given parton-level observable defines a multiplicative enhancement on top of the normal matrix element shape. Example:

```
- 93 93 -> 11 -11 93{1}:
  2->3:
    Enhance_Function: VAR{PPerp2(p[2]+p[3])/400}
```

In this example, Z+1-jet events with $p_\perp(Z) = 20$ GeV and Z+0-jet events will come with no enhancement, while other Z+1-jet events will be enhanced with $(p_\perp(Z)/20)^2$. Note: if you would define the enhancement function without the normalisation to $1/20^2$, the Z+1-jet would come with a significant overall enhancement compared to the unenhanced Z+0-jet process, which would have a strong impact on the statistical uncertainty in the Z+0-jet region.

Optionally, a range can be specified over which the multiplicative biasing should be applied. The matching at the range boundaries will be smooth, i.e. the effective enhancement is frozen to its value at the boundaries. Example:

```
- 93 93 -> 11 -11 93{1}:
  2->3:
    Enhance_Function: VAR{PPerp2(p[2]+p[3])/400}|1.0|100.0
```

This implements again an enhancement with $(p_\perp(Z)/20)^2$ but only in the range of 20-200 GeV. As you can see, you have to be take into account the normalisation also in the range specification.

### Enhance_Observable

Specifies a phase-space dependent biasing of parton-level events (before showering). Events will be statistically flat in the given observable and range. An example would be:

```
- 93 93 -> 11 -11 93{1}:
  2->3:
    Enhance_Observable: VAR{log10(PPerp(p[2]+p[3]))}|1|3
```

Here, the 1-jet process is flattened with respect to the logarithmic transverse momentum of the lepton pair in the limits 1.0 (10 GeV) to 3.0 (1 TeV). For the calculation of the observable one can use any function available in the algebra interpreter (see *Interpreter*).

The matching at the range boundaries will be smooth, i.e. the effective enhancement is frozen to its value at the boundaries.

This can have unwanted side effects for the statistical uncertainty when used in a multi-jet merged sample, because the flattening is applied in each multiplicity separately, and also affects the relative selection weights of each sub-sample (e.g. 2-jet vs. 3-jet).

---

**Note:** The convergence of the Monte Carlo integration can be worse if enhance functions/observables are employed and therefore the integration can take significantly longer. The reason is that the default phase space mapping, which is constructed according to diagrammatic information from hard matrix elements, is not suited for event generation including enhancement. It must first be adapted, which, depending on the enhance function and the final state multiplicity, can be an intricate task.

If Sherpa cannot achieve an integration error target due to the use of enhance functions, it might be appropriate to locally redefine this error target, see *Integration_Error*.

---

## 5.7 Selectors

Sherpa provides the following selectors that set up cuts at the matrix element level:

- *Inclusive selectors*
- *One particle selectors*
- *Two particle selectors*
- *Decay selectors*
- *Particle dressers*

Some selectors modify the momenta and flavours of the set of final state particles. These selectors also take a list of subselectors which then act on the modified flavours and momenta. Details are explained in the respective selectors' description.

### 5.7.1 Inclusive selectors

The selectors listed here implement cuts on the matrix element level, based on event properties. The corresponding syntax is

```
SELECTORS:
  - [<keyword>, <parameter 1>, <parameter 2>, ...]
  - # other selectors ...
```

Parameters that accept numbers can also be given in a form that is understood by the internal algebra interpreter, see *Interpreter*. The selectors act on *all* particles in the event. Their respective keywords are

**[N, <kf>, <min value>, <max value>]** Minimum and maximum multiplicity of flavour `<kf>` in the final state.

**[PTmis, <min value>, <max value>]** Missing transverse momentum cut (at the moment only neutrinos are considered invisible)

**[ETmis, <min value>, <max value>]** Missing transverse energy cut (at the moment only neutrinos are considered invisible)

**[IsolationCut, <kf>, <dR>, <exponent>, <epsilon>, <optional: mass_max>]** Smooth cone isolation [Fri98], the parameters given are the flavour `<kf>` to be isolated against massless partons and the isolation cone parameters.

**[NJ, <N>, <algo>, <min value>, <max value>]** NJettiness from [SJW10], where `<algo>` specifies the jet finding algorithm to determine the hard jet directions and `<N>` is their multiplicity. `algo=kt|antikt|cambridge|siscone,PT:<ptmin>,R:<dR>[,[ETA:<etamax>,Y:<ymax>]]`

### 5.7.2 One particle selectors

The selectors listed here implement cuts on the matrix element level, based on single particle kinematics. The corresponding syntax is

```
SELECTORS:
  - [<keyword>, <flavour code>, <min value>, <max value>]
  - # other selectors ...
```

`<min value>` and `<max value>` are floating point numbers, which can also be given in a form that is understood by the internal algebra interpreter, see *Interpreter*. The selectors act on *all* possible particles with the given flavour. Their respective keywords are

**PT** transverse momentum cut

**ET** transverse energy cut

**Y** rapidity cut

**Eta** pseudorapidity cut

**PZIN** cut on the z-component of the momentum, acts on initial-state flavours only (commonly used in DIS analyses)

### 5.7.3 Two particle selectors

The selectors listed here implement cuts on the matrix element level, based on two particle kinematics. The corresponding is

```
SELECTORS:
  - [<keyword>, <flavour1 code>, <flavour2 code>, <min value>, <max value>]
  - # other selectors ...
```

<min value> and <max value> are floating point numbers, which can also be given in a form that is understood by the internal algebra interpreter, see *Interpreter*. The selectors act on *all* possible particles with the given flavour. Their respective keywords are

**Mass** invariant mass

**Q2** DIS-like virtuality

**PT2** pair transverse momentum

**MT2** pair transverse mass

**DY** rapidity separation

**DEta** pseudorapidity separation

**DPhi** azimuthal separation

**DR** angular separation (build from eta and phi)

**DR(y)** angular separation (build from y and phi)

**INEL** inelasticity, one of the flavours must be in the initial-state (commonly used in DIS analyses)

### 5.7.4 Decay selectors

The selectors listed here implement cuts on the matrix element level, based on particle decays, see *Decay* and *DecayOS*.

**DecayMass** Invariant mass of a decaying particle. The syntax is

```
- [DecayMass, <flavour code>, <min value>, <max value>]
```

**Decay** Any kinematic variable of a decaying particle. The syntax is

```
- [Decay(<expression>), <flavour code>, <min value>, <max value>]
```

where <expression> is an expression handled by the internal interpreter, see *Interpreter*.

**Decay2** Any kinematic variable of a pair of decaying particles. The syntax is

```
- [Decay2(<expression>), <flavour1 code>, <flavour2 code>, <min value>, <max␣
↪value>]
```

where <expression> is an expression handled by the internal interpreter, see *Interpreter*.

Particles are identified by flavour, i.e. the cut is applied on *all* decaying particles that match <flavour code>. <min value> and <max value> are floating point numbers, which can also be given in a format that is understood by the internal algebra interpreter, see *Interpreter*.

### 5.7.5 Particle dressers

### 5.7.6 Jet selectors

There are two different types of jet finders

**NJetFinder** k_T-type algorithm to select on a given number of jets

**FastjetFinder** Select on a given number of jets using FastJet algorithms

Their respective syntax and defaults are

```
SELECTORS:
- NJetFinder:
    N: 0
    PTMin: 0.0
    ETMin: 0.0
    R: 0.4
    Exp: 1
    EtaMax: None
    YMax: None
    MassMax: 0.0
- FastjetFinder:
    Algorithm: kt
    N: 0
    PTMin: 0.0
    ETMin: 0.0
    DR: 0.4
    f: 0.75         # Siscone f parameter
    EtaMax: None
    YMax: None
    Nb: -1
    Nb2: -1
```

Note that all parameters are optional. If they are not specified, their respective default values as indicated in the above snippet is used. However, at the very least the number of jets, N, should be specified to require a non-zero number of jets.

The NJetFinder allows to select for kinematic configurations with at least <N> jets that satisfy both, the <PTMin> and the <ETMin> minimum requirements and that are in a pseudo-rapidity region |eta|. The <Exp> (exponent) allows to apply a kt-algorithm (1) or an anti-kt algorithm (-1). As only massless partons are clustered by default, the <MassMax> allows to also include partons with a mass up to the specified values. This is useful e.g. in calculations with massive b-quarks which shall nonetheless satisfy jet criteria.

The second option FastjetFinder allows to use the FastJet plugin, through fjcore. It takes the following arguments: <Algorithm> can take the values kt,antikt,cambridge,siscone,eecambridge,jade, <N> is the minimum number of jets to be found, <PTMin> and <ETMin> are the minimum transverse momentum and/or energy, <DR> is the radial parameter. Optional arguments are: <f> (default 0.75, only relevant for the Siscone algorithm), <EtaMax> and <YMax> as maximal absolute (pseudo-)rapidity, <Nb> and <Nb2> set the number of required b-jets, where for the former both b and anti-b quarks are counted equally towards b-jets, while for the latter they are added with a relative sign as constituents, i.e. a jet containing b and anti-b is not tagged (default: -1, i.e. no b jets are required). Note that only <Algorithm>, <N> and <PTMin> are relevant for the lepton-lepton collider algorithms.

The selector `FastjetVeto` allows to use the FastJet plugin to apply jet veto cuts. Its syntax is identical to `FastjetFinder`.

The momenta and nodal values of the jets found with FastJet can also be used to calculate more elaborate selector criteria. The syntax of this selector is

```
- FastjetSelector:
    Expression: <expression>
    Algorithm: kt
    N: 0
    PTMin: 0.0
    ETMin: 0.0
    DR: 0.4
    f: 0.75
    EtaMax: None
    YMax: None
    BMode: 0
```

wherein `Algorithm` can take the values `kt,antikt,cambridge,siscone,eecambridge,jade`. In the algebraic `<expression>`, `MU_n2` (n=2..njet+1) signify the nodal values of the jets found and `p[i]` are their momenta. For details see *Scale setters*. For example, in lepton pair production in association with jets

```
- FastjetSelector:
    Expression: Mass(p[4]+p[5])>100
    Algorithm: antikt
    N: 2
    PTMin: 40
    ETMin: 0
    DR: 0.5
```

selects all phase space points where two anti-kt jets with at least 40 GeV of transverse momentum and an invariant mass of at least 100 GeV are found. The expression must calculate a boolean value. The `BMode` parameter, if specified different from its default 0, allows to use b-tagged jets only, based on the parton-level constituents of the jets. There are two options: With `BMode: 1` both b and anti-b quarks are counted equally towards b-jets, while for `BMode: 2` they are added with a relative sign as constituents, i.e. a jet containing b and anti-b is not tagged. Note that only `<epression>`, `<algorithm>`, `<n>` and `<ptmin>` are relevant when using the lepton-lepton collider algorithms.

### 5.7.7 Isolation selector

Instead of the simple `IsolationCut` (*Inclusive selectors*), you may also use the more flexible `Isolation_Selector` to require photons (or other particles) with a smooth cone isolation and additionally apply further criteria to them. Example:

```
SELECTORS:
- Isolation_Selector:
    Isolation_Particle: 22
    Rejection_Particles: [93]
    Isolation_Parameters:
      R: 0.1
      EMAX: 0.1
      EXP: 2
      PT: 0.
      Y: 2.7
    NMin: 2
    Remove_Nonisolated: true
    Subselectors:
```

(continues on next page)

(continued from previous page)

```
    - VariableSelector:
        Variable: PT
        Flavs: [22]
        Ranges:
        - [20, E_CMS]
        - [18, E_CMS]
        Ordering: [PT_UP]
    - [DR, 22, 22, 0.2, 10000.0 ]
    #for integration efficiency: m_yy >= sqrt(2 pTmin1 pTmin2 (1-cos dR))
    - [Mass, 22, 22, 3.7, E_CMS]
```

### 5.7.8 Universal selector

The universal selector is intended to implement non-standard cuts on the matrix element level. Its syntax is

```
SELECTORS:
- VariableSelector:
    Variable: <variable>
    Flavs: [<kf1>, ..., <kfn>]
    Ranges:
    - [<min1>, <max1>]
    - ...
    - [<minn>, <maxn>]
    Ordering: [<order1>, ..., <orderm>]
```

The `Variable` parameter defines the name of the variable to cut on. The keywords for available predefined can be figured by running Sherpa `SHOW_VARIABLE_SYNTAX: true`. Or alternatively, an arbitrary cut variable can be constructed using the internal interpreter, see *Interpreter*. This is invoked with the command `Calc(...)`. In the formula specified there you have to use place holders for the momenta of the particles: `p[0] ... p[n]` hold the momenta of the respective particles `kf1 ... kfn`. A list of available vector functions and operators can be found here *Interpreter*.

`<kf1>,..,` specify the PDG codes of the particles the variable has to be calculated from. The ranges `[<min>, <max>]` then define the cut regions.

If the `Ordering` parameter is not given, the order of cuts is determined internally, according to Sherpa's process classification scheme. This then has to be matched if you want to have different cuts on certain different particles in the matrix element. To do this, you should put enough (for the possible number of combinations of your particles) arbitrary ranges at first and run Sherpa with debugging output for the universal selector: `Sherpa 'FUNCTION_OUTPUT: {"Variable_Selector::Trigger": 15}'`. This will start to produce lots of output during integration, at which point you can interrupt the run (Ctrl-c). In the `Variable_Selector::Trigger(): {...}` output you can see, which particle combinations have been found and which cut range your selector has held for them (vs. the arbitrary range you specified). From that you should get an idea, in which order the cuts have to be specified.

If the fourth argument is given, particles are ordered before the cut is applied. Possible orderings are `PT_UP`, `ET_UP`, `E_UP`, `ETA_UP` and `ETA_DOWN`, (increasing p_T, E_T, E, eta, and decreasing eta). They have to be specified for each of the particles, separated by commas.

Examples

```
SELECTORS:
# two-body transverse mass
- VariableSelector:
    Variable: mT
    Flavs: [11, -12]
```

(continues on next page)

(continued from previous page)

```
    Ranges:
    - [50, E_CMS]

# cut on the pT of only the hardest lepton in the event
- VariableSelector:
    Variable: PT
    Flavs: 90
    Ranges:
    - [50, E_CMS]
    Ordering: [PT_UP]

# using bool operations to restrict eta of the electron to |eta| < 1.1 or
# 1.5 < |eta| < 2.5
- VariableSelector:
    Variable: Calc(abs(Eta(p[0]))<1.1||(abs(Eta(p[0]))>1.5&&abs(Eta(p[0]))<2.5))
    Flavs: 11
    Ranges:
    - [1, 1]  # NOTE: this means true for bool operations

# requesting opposite side tag jets in VBF
- VariableSelector:
    Variable: Calc(Eta(p[0])*Eta(p[1]))
    Flavs: [93, 93]
    Ranges:
    - [-100, 0]
    Ordering: [PT_UP, PT_UP]

# restricting electron+photon mass to be outside of [87.0,97.0]
- VariableSelector:
    Variable: Calc(Mass(p[0]+p[1])<87.0||Mass(p[0]+p[1])>97.0)
    Flavs: [11, 22]
    Ranges:
    - [1, 1]

# in ``Z[lepton lepton] Z[lepton lepton]``, cut on mass of lepton-pairs
# produced from Z's
- VariableSelector:
    Variable: m
    Flavs: [90, 90]
    # here we use knowledge about the internal ordering to cut only on the
    # correct lepton pairs
    Ranges:
    - [80, 100]
    - [0, E_CMS]
    - [0, E_CMS]
    - [0, E_CMS]
    - [0, E_CMS]
    - [80, 100]
```

### 5.7.9 Minimum selector

This selector can combine several selectors to pass an event if either those passes the event. It is mainly designed to generate more inclusive samples that, for instance, include several jet finders and that allows a specification later. The syntax is

```
SELECTORS:
- MinSelector:
    Subselectors:
    - <selector 1>
    - <selector 2>
    ...
```

The *Minimum selector* can be used if constructed with other selectors mentioned in this section

## 5.8 Integration

The following parameters are used to steer the integration:



### 5.8.1 INTEGRATION_ERROR

Specifies the relative integration error target.

### 5.8.2 INTEGRATOR

Specifies the integrator. The possible integrator types depend on the matrix element generator. In general users should rely on the default value and otherwise seek the help of the authors, see *Authors*. Within AMEGIC++ the options `AMEGIC: {INTEGRATOR: <type>, RS_INTEGRATOR: <type>}` can be used to steer the behaviour of the default integrator.

- `4`: building up the channels is achieved through respecting the peak structure given by the propagators. The algorithm works recursively starting from the initial state.
- `5`: this is an extension of option 4. In the case of competing peaks (e.g. a Higgs boson decaying into W+W-, which further decay), additional channels are produced to account for all kinematical configurations where one of the propagating particles is produced on its mass shell.
- `6`: in contrast to option 4 the algorithm now starts from the final state. The extra channels described in option 5 are produced as well. This is the default integrator if both beams are hadronic.
- `7`: Same as option `4` but with tweaked exponents. Optimised for the integration of real-subtracted matrix-elements. This is the default integrator when at least one of the beams is not hadronic.

In addition, a few ME-generator independent integrators have been implemented for specific processes:

- `Rambo`: RAMBO [KSE86]. Generates isotropic final states.
- `VHAAG`: Vegas-improved HAAG integrator [vHP02].
- `VHAAG_res`: is an integrator for a final state of a weak boson, decaying into two particles plus two or more jets based on HAAG [vHP02]. This integrator can be further configured using `VHAAG` sub-settings, i.e. `VHAAG: {<sub-setting>: <value>}`. The following sub-settings are available. `RES_KF` specifies the kf-code of the weak boson, the default is W (`24`). `RES_D1` and `RES_D2` define the positions of the Boson decay products within the internal naming scheme, where `2` is the position of the first outgoing particle. The defaults are `2` and `3`, respectively, which is the correct choice for all processes where the decay products are the only not strongly interacting final state particles.

### 5.8.3 VEGAS_MODE

Specifies the mode of the Vegas adaptive integration. `0` disables Vegas, `2` enables it (default).

### 5.8.4 FINISH_OPTIMIZATION

Specifies whether the full Vegas optimization is to be carried out. The two possible options are `true` (default) and `false`.

### 5.8.5 PSI

The sub-settings for the phase space integrator can be customised as follows:

```
PSI:
  <sub-setting>: <value>
  # more PSI settings ...
```

The following sub-settings exist:

**NMAX** The maximum number of points before cuts to be generated during integration. This parameter acts on a process-by-process basis.

**ITMIN** The minimum number of points used for every optimisation cycle. Please note that it might be increased automatically for complicated processes.

**ITMAX** The maximum number of points used for every optimisation cycle. Please note that for complicated processes the number given might be insufficient for a meaningful optimisation.

**ITMIN_BY_NODE** Same as `ITMIN`, but specified per node to allow tuning of integration performance in large-scale MPI runs.

**ITMAX_BY_NODE** Same as `ITMAX`, but specified per node to allow tuning of integration performance in large-scale MPI runs.

## 5.9 Hard decays

The handler for decays of particles produced in the hard scattering process (e.g. W, Z, top, Higgs) can be enabled and configured using the `HARD_DECAYS` collection of settings (and a small number of other other top-level settings). Which (anti)particles IDs should be treated as unstable is determined by the `PARTICLE_DATA:<id>:Stable` switch described in *Models*.

The syntax to configure `HARD_DECAYS` sub-settings is:

```
HARD_DECAYS:
  <sub-setting>: <value>
  # more sub-settings ...
  Channels:
    "<channel id>":
      <channel sub-setting>: <value>
      # more sub-settings for <channel>
    # more channels ...
```

The channel ID codes are of the form `a -> b c ...`. The particle IDs for the decay channels can be found in the decay table printed to screen during the run.

This decay module can also be used on top of NLO matrix elements, but it does not include any NLO corrections in the decay matrix elements themselves.

Note that the decay handler is an afterburner at the event generation level. It does not affect the calculation and integration of the hard scattering matrix elements. The cross section is thus unaffected during integration, and the branching ratios (if any decay channels have been disabled) are only taken into account for the event weights and cross section output at the end of event generation (if not disabled with the `HARD_DECAYS:Apply_Branching_Ratios` option, cf. below). Furthermore any cuts or scale definitions are not affected by decays and operate only on the inclusively produced particles before decays.

### 5.9.1 Status

This sub-setting to each channel defined in `HARD_DECAYS:Channels` allows to explicitly force or disable a decay channel. The status can take the following values:

**`Status: -1`** Decay channel is disabled and does not contribute to total width.

**`Status: 0`** Decay channel is disabled but contributes to total width.

**`Status: 1 (default)`** Decay channel is enabled.

**`Status: 2`** Decay channel is forced.

For example, to disable the hadronic decay channels of the W boson one would use:

```
HARD_DECAYS:
  Channels:
    "24 -> 2 -1":   { Status: 0 }
    "24 -> 4 -3":   { Status: 0 }
    "-24 -> -2 1": { Status: 0 }
    "-24 -> -4 3": { Status: 0 }
```

In the same way, the bottom decay mode of the Higgs could be forced using:

```
"25 -> 5 -5":   { Status: 2 }
```

Note that the ordering of the decay products in `<channel id>` is important and has to be identical to the ordering in the decay table printed to screen. It is also possible to request multiple forced decay channels (`Status: 2`) for the same particle, all other channels will then automatically be disabled.

### 5.9.2 Width

This option allows to overwrite the calculated partial width (in GeV) of a given decay channel, and even to add new inactive channels which contribute to the total width. This is useful to adjust the branching ratios, which are used for the relative contributions of different channels and also influence the cross section during event generation, as well as the total width which is used for the lineshape of the resonance.

An example to set (/add) the partial widths of the `H->ff`, `H->gg` and `H->yy` channels can be seen in the following. The values have been taken from LHCHXSWG):

```
PARTICLE_DATA:
  25:
    Mass: 125
    Width: 0.00407

HARD_DECAYS:
  Enabled: true
  Channels:
    "25 -> 5 -5":    { Width: 2.35e-3 }
    "25 -> 15 -15": { Width: 2.57e-4 }
    "25 -> 13 -13": { Width: 8.91e-7 }
    "25 -> 4 -4":    { Width: 1.18e-4 }
    "25 -> 3 -3":    { Width: 1.00e-6 }
    "25 -> 21 21":   { Width: 3.49e-4 }
    "25 -> 22 22":   { Width: 9.28e-6 }
```

Another example, setting the leptonic and hadronic decay channels of W and Z bosons to the PDG values, would be specified as follows:

```
HARD_DECAYS:
  Enabled: true
  Channels:
    "24 -> 2 -1":     { Width: 0.7041 }
    "24 -> 4 -3":     { Width: 0.7041 }
    "24 -> 12 -11":   { Width: 0.2256 }
    "24 -> 14 -13":   { Width: 0.2256 }
    "24 -> 16 -15":   { Width: 0.2256 }
    "-24 -> -2 1":    { Width: 0.7041 }
    "-24 -> -4 3":    { Width: 0.7041 }
    "-24 -> -12 11":  { Width: 0.2256 }
    "-24 -> -14 13":  { Width: 0.2256 }
    "-24 -> -16 15":  { Width: 0.2256 }
    "23 -> 1 -1":     { Width: 0.3828 }
    "23 -> 2 -2":     { Width: 0.2980 }
    "23 -> 3 -3":     { Width: 0.3828 }
    "23 -> 4 -4":     { Width: 0.2980 }
    "23 -> 5 -5":     { Width: 0.3828 }
    "23 -> 11 -11":   { Width: 0.0840 }
    "23 -> 12 -12":   { Width: 0.1663 }
    "23 -> 13 -13":   { Width: 0.0840 }
    "23 -> 14 -14":   { Width: 0.1663 }
    "23 -> 15 -15":   { Width: 0.0840 }
    "23 -> 16 -16":   { Width: 0.1663 }
```

### 5.9.3 HARD_SPIN_CORRELATIONS

Spin correlations between the hard scattering process and the following decay processes are enabled by default. If you want to disable them, e.g. for spin correlation studies, you can specify the option `HARD_SPIN_CORRELATIONS: 0`. This is a top-level setting as opposed to the other `HARD_DECAYS`-related settings.

### 5.9.4 Store_Results

The decay table and partial widths are calculated on-the-fly during the initialization phase of Sherpa from the given model and its particles and interaction vertices. To store these results in the `Results/Decays` directory, one has to specify `HARD_DECAYS: { Store_Results: 1 }`. In case existing decay tables are to be read in the same configuration should be done. Please note, that Sherpa will delete decay channels present in the read in results but not in the present model with present parameters by default. To prevent Sherpa from updating the decay table files accordingly specify `HARD_DECAYS: { Store_Results: 2 }`.

### 5.9.5 Result_Directory

Specifies the name of the directory where the decay results are to be stored. Defaults to the value of the top-level setting *RESULT_DIRECTORY*.

### 5.9.6 Set_Widths

The decay handler computes LO partial and total decay widths and generates decays with corresponding branching fractions, independently from the particle widths specified by `PARTICLE_DATA:<id>:Width`. The latter are relevant only for the core process and should be set to zero for all unstable particles appearing in the core-process final state. This guarantees on-shellness and gauge invariance of the core process, and subsequent decays can be handled by the afterburner. In constrast, `PARTICLE_DATA:<id>:Width` should be set to the physical width when unstable particles appear (only) as intermediate states in the core process, i.e. when production and decay are handled as a full process or using `Decay`/`DecayOS`. In this case, the option `HARD_DECAYS: { Set_Widths: true }` permits to overwrite the `PARTICLE_DATA:<id>:Width` values of unstable particles by the LO widths computed by the decay handler.

### 5.9.7 Apply_Branching_Ratios

By default (`HARD_DECAYS: { Apply_Branching_Ratios: true }`), weights for events which involve a hard decay are multiplied with the corresponding branching ratios (if decay channels have been disabled). This also means that the total cross section at the end of the event generation run already includes the appropriate BR factors. If you want to disable that, e.g. because you want to multiply with your own modified BR, you can set the option `{HARD_DECAYS: { Apply_Branching_Ratios: false }`.

### 5.9.8 Mass_Smearing

With the default of `HARD_DECAYS: { Mass_Smearing: 1 }` the kinematic mass of the unstable propagator is distributed according to a Breit-Wigner shape a posteriori. All matrix elements are still calculated in the narrow-width approximation with onshell particles. Only the kinematics are affected. To keep all intermediate particles onshell `{HARD_DECAYS: { Mass_Smearing: 0 }`.

### 5.9.9 Resolve_Decays

There are different options how to decide when a 1->2 process should be replaced by the respective 1->3 processes built from its decaying daughter particles.

**`Resolve_Decays:`** **`Threshold`** (default) Only when the sum of decay product masses exceeds the decayer mass.

**`Resolve_Decays:`** **`ByWidth`** As soon as the sum of 1->3 partial widths exceeds the 1->2 partial width.

**`Resolve_Decays:`** **`None`** No 1->3 decays are taken into account.

In all cases, one can exclude the replacement of a particle below a given width threshold using `Min_Prop_Width:` (default 0.0). Both settings are sub-settings of `HARD_DECAYS`:

```
HARD_DECAYS:
  Resolve_Decays: <mode>
  Min_Prop_Width: <threshold>
```

### 5.9.10 Decay_Tau

By default, the tau lepton is decayed by the hadron decay module, *Hadron decays*, which includes not only the leptonic decay channels but also the hadronic modes. If `Decay_Tau: true` is specified, the tau lepton will be decayed in the hard decay handler, which only takes leptonic and partonic decay modes into account. Note, that in this case the tau needs to also be set massive:

```
PARTICLE_DATA:
  15:
    Massive: true
HARD_DECAYS:
  Decay_Tau: true
```

### 5.9.11 Decay table integration settings

Three parameters can be used to steer the accuracy and time consumption of the calculation of the partial widths in the decay table: `Int_Accuracy: 0.01` specifies a relative accuracy for the integration. The corresponding target reference is either the given total width of the decaying particle (`Int_Target_Mode: 0`, default) or the calculated partial decay width (`Int_Target_Mode: 1`). The option `Int_NIter: 2500` can be used to change the number of points per integration iteration, and thus also the minimal number of points to be used in an integration. All decay table integration settings are sub-settings of `HARD_DECAYS`.

## 5.10 Parton showers

The following parameters are used to steer the shower setup.



### 5.10.1 SHOWER_GENERATOR

There are two shower generators in Sherpa, `Dire` (default) and `CSS`. See the module summaries in *Basic structure* for details about these showers.

Other shower modules are in principle supported and more choices will be provided by Sherpa in the near future. To list all available shower modules, the tag `SHOW_SHOWER_GENERATORS: 1` can be specified on the command line.

`SHOWER_GENERATOR: None` switches parton showering off completely. However, even in the case of strict fixed order calculations, this might not be the desired behaviour as, for example, then neither the METS scale setter, cf. *SCALES*, nor Sudakov rejection weights can be employed. To circumvent when using the Dire or CS Shower see *Sherpa Shower options*.

### 5.10.2 JET_CRITERION

This option uses the value for `SHOWER_GENERATOR` as its default. Correspondingly, the only natively supported options in Sherpa are `CSS` and `Dire`. The corresponding jet criterion is described in [HKSS09]. A custom jet criterion, tailored to a specific experimental analysis, can be supplied using Sherpa's plugin mechanism.

### 5.10.3 MASSIVE_PS

This option instructs Sherpa to treat certain partons as massive in the shower, which have been considered massless by the matrix element. The argument is a list of parton flavours, for example `MASSIVE_PS: [4, 5]`, if both c- and b-quarks are to be treated as massive.

### 5.10.4 MASSLESS_PS

When hard decays are used, Sherpa treats all flavours as massive in the parton shower. This option instructs Sherpa to treat certain partons as massless in the shower nonetheless. The argument is a list of parton flavours, for example `MASSLESS_PS: [1, 2, 3]`, if u-, d- and s-quarks are to be treated as massless.

### 5.10.5 Sherpa Shower options

Sherpa's default shower module is based on [SK08b]. A new ordering parameter for initial state splitters was introduced in [HKSS09] and a novel recoil strategy for initial state splittings was proposed in [HSS10]. While the ordering variable is fixed, the recoil strategy for dipoles with initial-state emitter and final-state spectator can be changed for systematics studies. Setting `CSS_KIN_SCHEME: 0` corresponds to using the recoil scheme proposed in [HSS10], while `CSS_KIN_SCHEME: 1` (default) enables the original recoil strategy. The lower cutoff of the shower evolution can be set via `CSS_FS_PT2MIN` and `CSS_IS_PT2MIN` for final and initial state shower, respectively. Note that this value is specified in GeV^2. Scale factors for the evaluation of the strong coupling in the parton shower are given by `CSS_FS_AS_FAC` and `CSS_IS_AS_FAC`. They multiply the ordering parameter, which is given in units of GeV^2.

Setting `CSS_MAXEM:` forces the CS Shower to truncate its evolution at the Nth emission. Note that in this case not all of the Sudakov weights might be computed correctly. On the other hand, the use of CS Shower in the METS scale setter is not affected, cf. *SCALES*.

### 5.10.6 CS Shower options

By default, only QCD splitting functions are enabled in the CS shower. If you also want to allow for photon splittings, you can enable them by using `CSS_EW_MODE: true`. Note, that if you have leptons in your matrix-element final state, they are by default treated by a soft photon resummation as explained in *QED corrections*. To avoid double counting, this has to be disabled as explained in that section.

The evolution variable of the CS shower can be changed using `CSS_EVOLUTION_SCHEME`. Two options are currently implemented, which correspond to transverse momentum ordering (option 0) and modified transverse momentum ordering (option 1). In addition, modified versions of these options (option 2 and option 3) are implemented, which take parton masses into account where applicable. The scale at which the strong coupling for gluon splitting into quarks is evaluated can be chosen with `CSS_SCALE_SCHEME`, where 0 (default) corresponds to the ordering parameter and 1 corresponds to invariant mass. Additionally, the CS shower allows to disable splittings at scales below the on-shell mass of heavy quarks. The upper limit for the corresponding heavy quark mass is set using `CSS_MASS_THRESHOLD`.

## 5.11 Multiple interactions

The basic MPI model is described in [SvZ87] while Sherpa's implementation details are discussed in [A+a].

The following parameters are used to steer the MPI setup:



### 5.11.1 MI_HANDLER

Specifies the MPI handler. The two possible values at the moment are `None` and `Amisic`.

### 5.11.2 MI ISR parameters

The following two parameters can be used to overwrite the *ISR parameters* in the context of multiple interactions: `MPI_PDF_SET`, `MPI_PDF_SET_VERSIONS`.

### 5.11.3 TURNOFF

Specifies the transverse momentum turnoff in GeV.

### 5.11.4 SCALE_MIN

Specifies the transverse momentum integration cutoff in GeV.

### 5.11.5 PROFILE_FUNCTION

Specifies the hadron profile function. The possible values are `Exponential`, `Gaussian` and `Double_Gaussian`. For the double gaussian profile, the relative core size and relative matter fraction can be set using *PROFILE_PARAMETERS*.

### 5.11.6 PROFILE_PARAMETERS

The potential parameters for hadron profile functions, see *PROFILE_FUNCTION*. For double gaussian profiles there are two parameters, corresponding to the relative core size and relative matter fraction.

### 5.11.7 REFERENCE_SCALE

Specifies the centre-of-mass energy at which the transverse momentum integration cutoff is used as is, see *SCALE_MIN*. This parameter should not be changed by the user. The default is `1800`, corresponding to Tevatron Run I energies.

### 5.11.8 RESCALE_EXPONENT

Specifies the rescaling exponent for fixing the transverse momentum integration cutoff at centre-of-mass energies different from the reference scale, see *SCALE_MIN*, *REFERENCE_SCALE*.

### 5.11.9 TURNOFF_EXPONENT

Specifies the rescaling exponent for fixing the transverse momentum turnoff at centre-of-mass energies different from the reference scale, see *TURNOFF*, *REFERENCE_SCALE*.

### 5.11.10 SIGMA_ND_FACTOR

Specifies the factor to scale the non-diffractive cross section calculated in the MPI initialisation.

### 5.11.11 MI_RESULT_DIRECTORY

Specifies the name of the directory where the MPI grid is stored. The default comprises the beam particles, their energies and the PDF used. In its default value, this information safeguards against using unsuitable grids for the current calculation.

### 5.11.12 MI_RESULT_DIRECTORY_SUFFIX

Supplements the default directory name for the MPI grid with a suffix.

## 5.12 Hadronization

The hadronization setup covers the fragmentation of partons into primordial hadrons as well as the decays of unstable hadrons into stable final states.



### 5.12.1 Fragmentation

#### Fragmentation models

The `FRAGMENTATION` parameter sets the fragmentation module to be employed during event generation.

- The default is `Ahadic`, enabling Sherpa's native hadronization model AHADIC++, based on the cluster fragmentation model introduced in [FW83], [Web84], [GM87], and [MW88] and implementing some modifications discussed in [WKS04].
- the hadronization can be disabled with the value `None`.
- To evaluate uncertainties stemming from the hadronization, Sherpa also provides an interface to the Lund string fragmentation in Pythia 6.4 [SMS06] by using the setting `Lund`. In this case, the standard Pythia switches `MSTJ`, `MSTP`, `MSTU`, `PARP`, `PARJ` and `PARU` can be used to steer the behaviour of the Lund string, see [SMS06]. They can be specified as a 2xN matrix:

```
FRAGMENTATION: Lund
MSTJ:
- [<number1>, <value1>]
- [<number2>, <value2>]
  ...
MSTP:
- [<number1>, <value1>]
  ...
```

### Hadron constituents

The constituent masses of the quarks and diquarks are given by

- `M_UP_DOWN` (0.3 GeV),
- `M_STRANGE` (0.4 GeV),
- `M_CHARM` (1.8 GeV), and
- `M_BOTTOM` (5.1 GeV).

The diquark masses are composed of the quark masses and some additional parameters,

with

- `M_DIQUARK_OFFSET` (0.3 GeV),
- `M_BIND_0` (0.12 GeV), and
- `M_BIND_1` (0.5 GeV).

### Hadron multiplets

For the selection of hadrons emerging in such cluster transitions and decays, an overlap between the cluster flavour content and the flavour part of the hadronic wave function is formed. This may be further modified by production probabilities, organised by multiplet and given by the parameters

- `MULTI_WEIGHT_R0L0_PSEUDOSCALARS` (default 1.0),
- `MULTI_WEIGHT_R0L0_VECTORS` (default 1.0),
- `MULTI_WEIGHT_R0L0_TENSORS2` (default 0.75),
- `MULTI_WEIGHT_R0L1_SCALARS` (default 0.0),
- `MULTI_WEIGHT_R0L1_AXIALVECTORS` (default 0.0),
- `MULTI_WEIGHT_R0L2_VECTORS` (default 0.0),
- `MULTI_WEIGHT_R0L0_N_1/2` (default 1.0),
- `MULTI_WEIGHT_R1L0_N_1/2` (default 0.0),
- `MULTI_WEIGHT_R2L0_N_1/2` (default 0.0),
- `MULTI_WEIGHT_R1_1L0_N_1/2` (default 0.0),
- `MULTI_WEIGHT_R0L0_DELTA_3/2` (default 0.25),

In addition, there is a suppression factors applied to meson singlets,

- `SINGLET_SUPPRESSION` (default 1.0).

For the latter, Sherpa also allows to redefine the mixing angles through parameters such as

- `Mixing_0+` (default -14.1/180*M_PI),
- `Mixing_1-` (default 36.4/180*M_PI),
- `Mixing_2+` (default 27.0/180*M_PI),
- `Mixing_3-` (default 0.5411),
- `Mixing_4+` (default 0.6283),

And finally, some modifiers are applied to individual hadrons:

- ETA_MODIFIER (default 0.12),
- ETA_PRIME_MODIFIER (default 1.0),

### Cluster transition to hadrons - flavour part

The phase space effects due to these masses govern to a large extent the flavour content of the non-perturbative gluon splittings at the end of the parton shower and in the decay of clusters. They are further modified by relative probabilities with respect to the production of up/down flavours through the parameters

- STRANGE_FRACTION (default 0.42),
- BARYON_FRACTION (default 1.0),
- CHARM_BARYON_MODIFIER (default 1.0),
- BEAUTY_BARYON_MODIFIER (default 1.0),
- P_{QS/P_{QQ}} (default 0.2),
- P_{SS/P_{QQ}} (default 0.04), and
- P_{QQ_1/P_{QQ_0}} (default 0.20).

The transition of clusters to hadrons is governed by the following considerations:

- Clusters can be interpreted as excited hadrons, with a continous mass spectrum.
- When a cluster becomes sufficiently light such that its mass is below the largest mass of any hadron with the same flavour content, it must be re-iterpreted as such a hadron. In this case it will be shifted on the corresponding hadron mass, and the recoil will be distributed to the "neighbouring" clusters or by emitting a soft photon. This comparison of masses clearly depends on the multiplets switched on in AHADIC++.
- In addition, clusters may becomes sufficiently light such that they should decay directly into two hadrons instead of two clusters. This decision is based on the heaviest hadrons accessible in a decay, modulated by another offset parameter,
  - DECAY_THRESHOLD (default 500 MeV).
- If both options, transition and decay, are available, there is a competition between

### Cluster transition and decay weights

The probability for a cluster C to be transformed into a hadron H is given by a combination of weights, obtained from the overlap with the flavour part of the hadronic wave function, the relative weight of the corresponding multiplet and a kinematic weight taking into account the mass difference of cluster and hadron and the width of the latter.

For the direct decay of a cluster into two hadrons the overlaps with the wave functions of all hadrons, their respective multiplet suppression weights, the flavour weight for the creation of the new flavour q and a kinematical factor are relevant. Here, yet another tuning paramter enters,

- MASS_EXPONENT (default 4.0)

which partially compensates phase space effects favouring light hadrons,

#### Cluster decays - kinematics

Cluster decays are generated by firstly emitting a non-perturbative "gluon" from one of the quarks, using a transverse momentum distribution as in the non-perturbative gluon decays, see below, and by then splitting this gluon into a quark–antiquark of anti-diquark–diquark pair, again with the same kinematics. In the first of these splittings, the emission of the gluon, though, the energy distribution of the gluon is given by the quark splitting function, if this quark has been produced in the perturbative phase of the event. If, in contrast, the quark stems from a cluster decay, the energy of the gluon is selected according to a flat distribution.

In clusters decaying to hadrons, the transverse momentum is chosen according to a distribution given by an infrared-continued strong coupling and a term inversemly proportional to the infrared-modified transverse momentum,

constrained to be below a maximal transverse momentum.

#### Splitting kinematics

In each splitting, the kinematics is given by the transverse momentum, the energy splitting parameter and the azimuthal angle. The latter, the azimuthal angle is always seleectred according to a flat distribution, while the energy splitting parameter will either be chosen according to the quark-to-gluon splitting function (if the quark is a leading quark, i.e. produced in the pertrubative phase), to the gluon-to-quark splitting function, or according to a flat distribution. The transverse momentum is given by the same distribution as in the cluster decays to hadrons.

### 5.12.2 Hadron decays

The treatment of hadron and tau decays is specified by `DECAYMODEL`. Its allowed values are either the default choice `Hadrons` (the default if `FRAGMENTATION: Ahadic`), `Lund` (the default if `FRAGMENTATION: Lund`), or it can be disabled with the option `Off`.

HADRONS++ is the module within the Sherpa framework which is responsible for treating hadron and tau decays. It contains decay tables with branching ratios for approximately 2500 decay channels, of which many have their kinematics modelled according to a matrix element with corresponding form factors. Especially decays of the tau lepton and heavy mesons have form factor models similar to dedicated codes like Tauola [JWDK93] and EvtGen [Lan01].

Some general switches which relate to hadron decays are

- `DECAYPATH` The path to the parameter files for the hadron and tau decays (default: `Decaydata/`). It is important to note that the path has to be given relative to the current working directory. If it doesn't exist, the default Decaydata directory (`<prefix>/share/SHERPA-MC/Decaydata`) will be used.
- Hadron properties like mass, width, stable/unstable and active can be set in full analogy to the settings for fundamental particles using `PARTICLE_DATA`, cf. *Models*.
- `SOFT_MASS_SMEARING = [0,1,2]` (default: 1) Determines whether particles entering the hadron decay event phase should be put off-shell according to their mass distribution. It is taken care that no decay mode is suppressed by a potentially too low mass. While HADRONS++ determines this dynamically from the chosen decay channel, for `Pythia` as hadron decay handler its `w-cut` parameter is employed. Choosing option 2 instead of 1 will only set unstable (decayed) particles off-shell, but leave stable particles on-shell.
- `MAX_PROPER_LIFETIME = [mm]` Parameter for maximum proper lifetime (in mm) up to which particles are considered unstable. If specified, this will make long-living particles stable, even if they are set unstable by default or by the user.

Many aspects of the above mentioned "Decaydata" can be adjusted. There exist three levels of data files, which are explained in the following sections. As with all other setup files, the user can either employ the default "Decaydata" in `<prefix>/share/SHERPA-MC/Decaydata`, or overwrite it (also selectively) by creating the appropriate files in the directory specified by `DECAYPATH`.

### HadronDecays.dat

`HadronDecays.dat` consists of a table of particles that are to be decayed by HADRONS++. Note: Even if decay tables exist for the other particles, only those particles decay that are set unstable, either by default, or in the model/fragmentation settings. It has the following structure, where each line adds one decaying particle:

| <kf-code> | <subdirectory> | <filename>.dat |
|---|---|---|
| decaying particle | path to decay table | decay table file |
| default names: | <particle>/ | Decays.dat |

It is possible to specify different decay tables for the particle (positive kf-code) and anti-particle (negative kf-code). If only one is specified, it will be used for both particle and anti-particle.

If more than one decay table is specified for the same kf-code, these tables will be used in the specified sequence during one event. The first matching particle appearing in the event is decayed according to the first table, and so on until the last table is reached, which will be used for the remaining particles of this kf-code.

Additionally, this file may contain the keyword `CREATE_BOOKLET` on a separate line, which will cause HADRONS++ to write a LaTeX document containing all decay tables.

### Decay table files

The decay table contains information about outgoing particles for each channel, its branching ratio and eventually the name of the file that stores parameters for a specific channel. If the latter is not specified HADRONS++ will produce it and modify the decay table file accordingly.

Additionally to the branching ratio, one may specify the error associated with it, and its source. Every hadron is supposed to have its own decay table in its own subdirectory. The structure of a decay table is

| {kf1,kf2,kf3,...} | BR(delta BR)[Origin] | <filename>.dat |
|---|---|---|
| outgoing particles | branching ratio | decay channel file |

It should be stressed here that the branching ratio which is explicitly given for any individual channel in this file is **always used** regardless of any matrix-element value.

### Decay channel files

A decay channel file contains various information about that specific decay channel. There are different sections, some of which are optional:

- ```
  <Options>
      AlwaysIntegrate = 0
      CPAsymmetryC = 0.0
      CPAsymmetryS = 0.0
  </Options>
  ```

  - `AlwaysIntegrate = [0,1]` For each decay channel, one needs an integration result for unweighting the kinematics (see below). This result is stored in the decay channel file, such that the integration is not needed for each run. The AlwaysIntegrate option allows to bypass the stored integration result, and do the integration nonetheless (same effect as deleting the integration result).
  - `CPAsymmetryC/CPAsymmetryS` If one wants to include time dependent CP asymmetries through interference between mixing and decay one can set the coefficients of the cos and sin terms respectively.
  ```

HADRONS++ will then respect these asymmetries between particle and anti-particle in the choice of decay channels.

- ```
  <Phasespace>
    1.0 MyIntegrator1
    0.5 MyIntegrator2
  </Phasespace>
  ```

  Specifies the phase-space mappings and their weight.

- ```
  <ME>
    1.0 0.0 my_matrix_element[X,X,X,X,X,...]
    1.0 0.0 my_current1[X,X,...] my_current2[X,X,X,...]
  </ME>
  ```

  Specifies the matrix elements or currents used for the kinematics, their respective weights, and the order in which the particles (momenta) enter them. For more details, the reader is referred to [KLS].

- ```
  <my_matrix_element[X,X,X,X,X,...]>
    parameter1 = value1
    parameter2 = value2
    ...
  </my_matrix_element[X,X,X,X,X,...]>
  ```

  Each matrix element or current may have an additional section where one can specify needed parameters, e.g. which form factor model to choose. Each parameter has to be specified on a new line as shown above. Available parameters are listed in [KLS]. Parameters not specified get a default value, which might not make sense in specific decay channels. One may also specify often needed parameters in `HadronConstants.dat`, but they will get overwritten by channel specific parameters, should these exist.

- ```
  <Result>
    3.554e-11 6.956e-14 1.388e-09;
  </Result>
  ```

  These last three lines have quite an important meaning. If they are missing, HADRONS++ integrates this channel during the initialization and adds the result lines. If this section exists though, and `AlwaysIntegrate` is off (the default value, see above) then HADRONS++ reads in the maximum for the kinematics unweighting.

  Consequently, if some parameters are changed (also masses of incoming and outgoing particles) the maximum might change such that a new integration is needed in order to obtain correct kinematical distributions. There are two ways to enforce the integration: either by deleting the last three lines or by setting `AlwaysIntegrate` to 1. When a channel is re-integrated, HADRONS++ copies the old decay channel file into `.<filename>.dat.old`.

### HadronConstants.dat

`HadronConstants.dat` may contain some globally needed parameters (e.g. for neutral meson mixing, see [KLS]) and also fall-back values for all matrix-element parameters which one specifies in decay channel files. Here, the `Interference_X = 1` switch would enable rate asymmetries due to CP violation in the interference between mixing and decay (cf. *Decay channel files*), and setting `Mixing_X = 1` enables explicit mixing in the event record according to the time evolution of the flavour states. By default, all mixing effects are turned off.

Mixing parameters with some example values

```
x_K = 0.946
y_K = -0.9965
```

(continues on next page)

(continued from previous page)

```
qoverp2_K = 1.0
Interference_K = 0
Mixing_K = 0

x_D = 0.0
y_D = 0.0
qoverp2_D = 1.0
Interference_D = 0
Mixing_D = 0

x_B = 0.776
y_B = 0.0
qoverp2_B = 1.0
Interference_B = 1
Mixing_B = 0

x_B(s) = 30.0
y_B(s) = 0.155
qoverp2_B(s) = 1.0
Interference_B(s) = 0
Mixing_B(s) = 0
```

### Further remarks

**Spin correlations:** a spin correlation algorithm is implemented. It can be switched on through the setting `SOFT_SPIN_CORRELATIONS: 1`.

If spin correlations for tau leptons produced in the hard scattering process are supposed to be taken into account, one needs to specify `HARD_SPIN_CORRELATIONS: 1` as well. If using AMEGIC++ as ME generator, note that the Process libraries have to be re-created if this is changed.

**Adding new channels:** if new channels are added to HADRONS++ (choosing isotropic decay kinematics) a new decay table must be defined and the corresponding hadron must be added to `HadronDecays.dat`. The decay table merely needs to consist of the outgoing particles and branching ratios, i.e. the last column (the one with the decay channel file name) can safely be dropped. By running Sherpa it will automatically produce the decay channel files and write their names in the decay table.

**Some details on tau decays:** $\tau$ decays are treated within the HADRONS++ framework, even though the $\tau$ is not a hadron. As for many hadron decays, the hadronic tau decays have form factor models implemented, for details the reader is referred to [KLS].

## 5.13 QED corrections

Higher order QED corrections are effected both on hard interaction and, upon their formation, on each hadron's subsequent decay. The Photons [SK08a] module is called in both cases for this task. It employes a YFS-type resummation [YFS61] of all infrared singular terms to all orders and is equipped with complete first order corrections for the most relevant cases (all other ones receive approximate real emission corrections built up by Catani-Seymour splitting kernels).

### 5.13.1 General Switches

The relevant switches to steer the higher order QED corrections are collected in the `YFS` settings group and are modified like this:

```
YFS:
  <option1>: <value1>
  <option2>: <value2>
  ...
```

The options are



#### MODE

The keyword `MODE` determines the mode of operation of Photons. `MODE: None` switches Photons off. Consequently, neither the hard interaction nor any hadron decay will be corrected for soft or hard photon emission. `MODE: Soft` sets the mode to "soft only", meaning soft emissions will be treated correctly to all orders but no hard emission corrections will be included. With `MODE: Full` these hard emission corrections will also be included up to first order in alpha_QED. This is the default setting.

#### USE_ME

The switch `USE_ME` tells Photons how to correct hard emissions to first order in alpha_QED. If `USE_ME: 0`, then Photons will use collinearly approximated real emission matrix elements. Virtual emission matrix elements of order alpha_QED are ignored. If, however, YFS_USE_ME=1, then exact real and/or virtual emission matrix elements are used wherever possible. These are presently available for V->FF, V->SS, S->FF, S->SS, S->Slnu, S->Vlnu type decays, Z->FF decays and leptonic tau and W decays. For all other decay types general collinearly approximated matrix elements are used. In both approaches all hadrons are treated as point-like objects. The default setting is `USE_ME: 1`. This switch is only effective if `MODE: Full`.

#### IR_CUTOFF

`IR_CUTOFF` sets the infrared cut-off dividing the real emission in two regions, one containing the infrared divergence, the other the "hard" emissions. This cut-off is currently applied in the rest frame of the multipole of the respective decay. It also serves as a minimum photon energy in this frame for explicit photon generation for the event record. In contrast, all photons below with energy less than this cut-off will be assumed to have negligible impact on the final-state momentum distributions. The default is `IR_CUTOFF: 1E-3` (GeV). Of course, this switch is only effective if Photons is switched on, i.e. `MODE` is not set to `None`.

### 5.13.2 QED Corrections to the Hard Interaction

The switches to steer QED corrections to the hard scattering are collected in the `ME_QED` settings group and are modified like this:

```
ME_QED:
  <option1>: <value1>
  <option2>: <value2>
  ...
```

The following options can be customised:



#### ENABLED

`ENABLED: false` turns the higher order QED corrections to the matrix element off. The default is `true`. Switching QED corrections to the matrix element off has no effect on *QED Corrections to Hadron Decays*. The QED corrections to the matrix element will only be effected on final state not strongly interacting particles. If a resonant production subprocess for an unambiguous subset of all such particles is specified via the process declaration (cf. *Processes*) this can be taken into account and dedicated higher order matrix elements can be used (if `YFS: { MODE: Full, USE_ME: 1 }`).

#### CLUSTERING_ENABLED

`CLUSTERING_ENABLED: false` switches the phase space point dependent identification of possible resonances within the hard matrix element on or off, respectively. The default is `true`. Resonances are identified by recombining the electroweak final state of the matrix element into resonances that are allowed by the model. Competing resonances are identified by their on-shell-ness, i.e. the distance of the decay product's invariant mass from the nominal resonance mass in units of the resonance width.

#### CLUSTERING_THRESHOLD

Sets the maximal distance of the decay product invariant mass from the nominal resonance mass in units of the resonance width in order for the resonance to be identified. The default is `CLUSTERING_THRESHOLD: 10.0`.

### 5.13.3 QED Corrections to Hadron Decays

If the Photons module is switched on, all hadron decays are corrected for higher order QED effects.

## 5.14 Minimum bias events

Minimum bias events are simulated through the Shrimps module in Sherpa.

### 5.14.1 Physics of Shrimps

#### Inclusive part of the model

Shrimps is based on the KMR model [RMK09], which is a multi-channel eikonal model. The incoming hadrons are written as a superposition of Good-Walker states, which are diffractive eigenstates that diagalonise the T-matrix. This allows to include low-mass diffractive excitation. Each combination of colliding Good-Walker states gives rise to a single-channel eikonal. The final eikonal is the superposition of the single-channel eikonals. The number of Good-Walker states is 2 in Shrimps (the original KMR model includes 3 states).

Each single-channel eikonal can be seen as the product of two parton densities, one from each of the colliding Good-Walker states. The evolution of the parton densities in rapidity due to extra emissions and absoption on either of the two hadrons is described by a set of coupled differential equations. The parameter `Delta`, which can be interpreted as the Pomeron intercept, is the probability for emitting an extra parton per unit of rapidity. The strength of absorptive corrections is quantified by the parameter `lambda`, which can also be seen as the triple-Pomeron coupling. A small region of size `deltaY` around the beams is excluded from the evolution due to the finite longitudinal size of the parton densities.

The boundary conditions for the parton densities are form factors, which have a dipole form characterised by the parameters `Lambda2`, `beta_02(mb)`, `kappa` and `xi`.

In this framework the eikonals and the cross sections for the different modes (elastic, inelastic, single- and double-diffractive) are calculated.

#### Exclusive part of the model

The description of this part of the model is outdated and needs to be updated. Please contact the *Authors* if you need more information.

### 5.14.2 Parameters and settings

Below is a list of all relevant parameters to steer the Shrimps module.

#### Generating minimum bias events

To generate minimum bias events with Shrimps `EVENT_TYPE` has to be set to `MinimumBias` and `SOFT_COLLISIONS` to `Shrimps`.

#### Shrimps Mode

The setup of minimum bias events is done via top-level settings. The exact choice is steered through the parameter `Shrimps_Mode` (default `Inelastic`), which allows the following settings:

- `Xsecs`, which will only calculate total, elastic, inelastic, single- and double-diffractive cross sections at various relevant energies and write them to a file, typically 'InclusiveQuantities/Xsecs.dat';
- `Elastic` generates elastic events at a fixed energy;
- `Single-diffractive` generates low-mass single-diffractive events at a fixed energy, modelled by the transition of one of the protons to a N(1440) state;
- `Double-diffractive` generates low-mass single-diffractive events at a fixed energy, modelled by the transition of both protons to N(1440) states;
- `Quasi-elastic` generates a combination of elastic, single- and double-diffractive events in due proportion;
- `Inelastic` generates inelastic minimum bias events through the exchange of t-channel gluons or singlets (pomerons). This mode actually will include large mass diffraction;
- `All` generates a combination of quasi-elastic and inelastic events in due proportion.

#### Parameters of the eikonals

The parameters of the differential equations for the parton densities are

- `Delta` (default 0.3): perturbative Pomeron intercept
- `lambda` (default 0.5): triple Pomeron coupling
- `deltaY` (default 1.5): rapidity interval excluded from evolution

The form factors are of the form:

$$F_{1/2}(q_T) = \beta_0^2(1 \pm \kappa)\frac{\exp(\frac{-\xi(1\pm\kappa)q_T^2}{\Lambda^2})}{(1+(1\pm\kappa)q_T^2/\Lambda^2)^2}$$

with the parameters

- $\Lambda^2$ (default 1.7 GeV^2)
- $\beta_0^2(mb)$ (default 25.0 mb)
- $\kappa$ (default 0.6)
- $\xi$ (default 0.2)

### Parameters for event generation

The description of these parameters is outdated and needs to be updated. Please contact the *Authors* if you need more information.

CHAPTER

SIX

# TIPS AND TRICKS



## 6.1 Shell completion

Sherpa will install a file named `$prefix/share/SHERPA-MC/sherpa-completion` which contains tab completion functionality for the bash shell. You simply have to source it in your active shell session by running

```
$ . $prefix/share/SHERPA-MC/sherpa-completion
```

and you will be able to tab-complete any parameters on a Sherpa command line.

To permanently enable this feature in your bash shell, you'll have to add the source command above to your `~/.bashrc`.

## 6.2 Rivet analyses

Sherpa is equipped with an interface to the analysis tool Rivet. To enable it, Rivet and HepMC have to be installed (e.g. using the Rivet bootstrap script) and your Sherpa compilation has to be configured with the following options:

```
$ ./configure --enable-hepmc2=/path/to/hepmc2 --enable-rivet=/path/to/rivet
```

(Note: Both paths are equal if you used the Rivet bootstrap script.)

To use the interface, you need to enable it using the `ANALYSIS` option and to configure it it using the `RIVET` settings group as follows:

```
ANALYSIS: Rivet
RIVET:
  ANALYSES:
    - D0_2008_S7662670
    - CDF_2007_S7057202
    - D0_2004_S5992206
    - CDF_2008_S7828950
```

The `ANALYSES` list specifies which Rivet analyses to run and the histogram output file can be changed with the normal `ANALYSIS_OUTPUT` switch.

You can also use `rivet-mkhtml` (distributed with Rivet) to create plot webpages from Rivet's output files:

```
$ source /path/to/rivetenv.sh   # see below
$ rivet-mkhtml -o output/ file1.yoda [file2.yoda, ...]
$ firefox output/index.html &
```

If your Rivet installation is not in a standard location, the bootstrap script should have created a `rivetenv.sh` which you have to source before running the `rivet-mkhtml` script.

## 6.3 HZTool analyses

Sherpa is equipped with an interface to the analysis tool HZTool. To enable it, HZTool and CERNLIB have to be installed and your Sherpa compilation has to be configured with the following options:

```
$ ./configure --enable-hztool=/path/to/hztool --enable-cernlib=/path/to/cernlib --
↪enable-hepevtsize=4000
```

Note that an example CERNLIB installation bootstrap script is provided in `AddOns/HZTool/start_cern_64bit`. Note that this script is only provided for convenience, we will not provide support if it is not working as expected.

To use the interface, enable it using the `ANALYSIS` and configure it using the `HZTool` settings group:

```
ANALYSIS: HZTool
HZTOOL:
  HISTO_NAME: output.hbook
  HZ_ENABLE:
  - hz00145
  - hz01073
  - hz02079
  - hz03160
```

The HZ_ENABLE list specifies which HZTool analyses to run. The histogram output directory can be changed using the ANALYSIS_OUTPUT switch, while HZTOOL:HISTO_NAME specifies the hbook output file.

## 6.4 MCFM interface

Sherpa is equipped with an interface to the NLO library of MCFM for decdicated processes. To enable it, MCFM has to be installed and compiled into a single library, libMCFM.a. To this end, an installation script is provided in AddOns/MCFM/install_mcfm.sh. Please note, due to some process specific changes that are made by the installation script to the MCFM code, only few selected processes of MCFM-6.3 are available through the interface.

Finally, your Sherpa compilation has to be configured with the following options:

```
$ ./configure --enable-mcfm=/path/to/mcfm
```

To use the interface, specify

```
Loop_Generator: MCFM
```

in the process section of the run card and add it to the list of generators in *ME_GENERATORS*. Of course, MCFM's process.DAT file has to be copied to the current run directory.

## 6.5 Debugging a crashing/stalled event

### 6.5.1 Crashing events

If an event crashes, Sherpa tries to obtain all the information needed to reproduce that event and writes it out into a directory named

```
Status__<date>_<time>
```

If you are a Sherpa user and want to report this crash to the Sherpa team, please attach a tarball of this directory to your email. This allows us to reproduce your crashed event and debug it.

To debug it yourself, you can follow these steps (Only do this if you are a Sherpa developer, or want to debug a problem in an addon library created by yourself):

- Copy the random seed out of the status directory into your run path:

  ```
  $ cp  Status__<date>_<time>/random.dat  ./
  ```

- Run your normal Sherpa commandline with an additional parameter:

  ```
  $ Sherpa [...] 'STATUS_PATH: ./'
  ```

  Sherpa will then read in your random seed from "./random.dat" and generate events from it.

- Ideally, the first event will lead to the crash you saw earlier, and you can now turn on debugging output to find out more about the details of that event and test code changes to fix it:

  ```
  $ Sherpa [...] --output 15 'STATUS_PATH: ./'
  ```

### 6.5.2 Stalled events

If event generation seems to stall, you first have to find out the number of the current event. For that you would terminate the stalled Sherpa process (using Ctrl-c) and check in its final output for the number of generated events. Now you can request Sherpa to write out the random seed for the event before the stalled one:

```
$ Sherpa [...] --events <#events - 1> 'SAVE_STATUS: Status/'
```

(Replace <#events - 1> using the number you figured out earlier.)

The created status directory can either be sent to the Sherpa developers, or be used in the same steps as above to reproduce that event and debug it.

## 6.6 Versioned installation

If you want to install different Sherpa versions into the same prefix (e.g. */usr/local*), you have to enable versioning of the installed directories by using the configure option `--enable-versioning`. Optionally you can even pass an argument to this parameter of what you want the version tag to look like.

## 6.7 NLO calculations



### 6.7.1 Choosing DIPOLES ALPHA

A variation of the parameter `DIPOLES:ALPHA` (see *Dipole subtraction*) changes the contribution from the real (subtracted) piece (`RS`) and the integrated subtraction terms (`I`), keeping their sum constant. Varying this parameter provides a nice check of the consistency of the subtraction procedure and it allows to optimize the integration performance of the real correction. This piece has the most complicated momentum phase space and is often the most time consuming part of the NLO calculation. The optimal choice depends on the specific setup and can be determined best by trial.

Hints to find a good value:

- The smaller `DIPOLES:ALPHA` is the less dipole term have to be calculated, thus the less time the evaluation/phase space point takes.
- Too small choices lead to large cancelations between the `RS` and the `I` parts and thus to large statisical errors.
- For very simple processes (with only a total of two partons in the iniatial and the final state of the born process) the best choice is typically `DIPOLES: {ALPHA: 1}`. The more complicated a process is the smaller `DIPOLES:ALPHA` should be (e.g. with 5 partons the best choice is typically around 0.01).
- A good choice is typically such that the cross section from the `RS` piece is significantly positive but not much larger than the born cross section.

### 6.7.2 Integrating complicated Loop-ME

For complicated processes the evaluation of one-loop matrix elements can be very time consuming. The generation time of a fully optimized integration grid can become prohibitively long. Rather than using a poorly optimized grid in this case it is more advisable to use a grid optimized with either the born matrix elements or the born matrix elements and the finite part of the integrated subtraction terms only, working under the assumption that the distibutions in phase space are rather similar.

This can be done by one of the following methods:

1. Employ a dummy virtual (requires no computing time, returns a finite value as its result) to optimise the grid. This only works if `V` is not the only `NLO_Part` specified.
    1. During integration set the `Loop_Generator` to `Dummy`. The grid will then be optimised to the phase space distribution of the sum of the Born matrix element and the finite part of the integrated subtraction term, plus a finite value from `Dummy`.

       **Note:** The cross section displayed during integration will also correspond to these contributions.

    2. During event generation reset `Loop_Generator` to your generator supplying the virtual correction. The events generated then carry the correct event weight.
2. Suppress the evaluation of the virtual and/or the integrated subtraction terms. This only works if Amegic is used as the matrix element generator for the `BVI` pieces and `V` is not the only `NLO_Part` specified.
    1. During integration add `AMEGIC: { NLO_BVI_MODE: <num> }` to your configuration. `<num>` takes the following values: `1-B`, `2-I`, and `4-V`. The values are additive, i.e. `3-BI`.

       **Note:** The cross section displayed during integration will match the parts selected by `NLO_BVI_MODE`.

    2. During event generation remove the switch again and the events will carry the correct weight.

**Note:** this will not work for the `RS` piece!

### 6.7.3 Avoiding misbinning effects

Close to the infrared limit, the real emission matrix element and corresponding subtraction events exhibit large cancellations. If the (minor) kinematics difference of the events happens to cross a parton-level cut or analysis histogram bin boundary, then large spurious spikes can appear.

These can be smoothed to some extend by shifting the weight from the subtraction kinematic to the real-emission kinematic if the dipole measure alpha is below a given threshold. The fraction of the shifted weight is inversely proportional to the dipole measure, such that the final real-emission and subtraction weights are calculated as:

```
w_r -> w_r + sum_i [1-x(alpha_i)] w_{s,i}
foreach i: w_{s,i} -> x(alpha_i) w_{s,i}
```

with the function $x(\alpha) = (\frac{\alpha}{|\alpha_0|})^n$ for $\alpha < \alpha_0$ and 1 otherwise.

The threshold can be set by the parameter `NLO_SMEAR_THRESHOLD: <alpha_0>` and the functional form of alpha and thus interpretation of the threshold can be chosen by its sign (positive: relative dipole kT in GeV, negative: dipole alpha). In addition, the exponent n can be set by `NLO_SMEAR_POWER: <n>`.

### 6.7.4 Enforcing the renormalization scheme

Sherpa takes information about the renormalization scheme from the loop ME generator. The default scheme is MSbar, and this is assumed if no loop ME is provided, for example when integrated subtraction terms are computed by themselves. This can lead to inconsistencies when combining event samples, which may be avoided by setting `AMEGIC: { LOOP_ME_INIT: 1 }`.

### 6.7.5 Checking the pole cancellation

The following options are all sub-settings for `AMEGIC` and can be specified as follows:

```
AMEGIC:
  <option>: <value>
  ...
```

To check whether the poles of the dipole subtraction and the interfaced one-loop matrix element cancel phase space point by phase space point `CHECK_POLES: 1` can be specified. In the same way, the finite contributions of the infrared subtraction and the one-loop matrix element can be checked by setting `CHECK_FINITE: 1`, and the Born matrix element via `CHECK_BORN: 1`. The accuracy to which the poles, finite parts and Born matrix elements are checked is set via `CHECK_THRESHOLD: <accu>`.

CHAPTER

SEVEN

# A POSTERIORI SCALE VARIATIONS

There are several ways to compute the effects of changing the scales and PDFs of any event produced by Sherpa. They can computed explicitly, cf. *Explicit scale variations*, on-the-fly, cf. *Scale and PDF variations* (restricted to multiplicative factors), or reconstructed a posteriori. The latter method needs plenty of additional information in the event record and is (depending on the actual calculation) available in two formats:



## 7.1 A posteriori scale and PDF variations using the HepMC GenEvent Output

Events generated in a LO, LOPS, NLO, NLOPS, MEPS@LO, MEPS@NLO or MENLOPS calculation can be written out in the HepMC format including all infomation to carry out arbitrary scale variations a posteriori. For this feature HepMC of at least version 2.06 is necessary and both `HEPMC_USE_NAMED_WEIGHTS: true` and `HEPMC_EXTENDED_WEIGHTS: true` have to enabled. Detailed instructions on how to use this information to construct the new event weight can be found here https://sherpa.hepforge.org/doc/ScaleVariations-Sherpa-2.2.0.pdf.

## 7.2 A posteriori scale and PDF variations using the ROOT NTuple Output

Events generated at fixed-order LO and NLO can be stored in ROOT NTuples that allow arbitrary a posteriori scale and PDF variations, see *Event output formats*. An example for writing and reading in such ROOT NTuples can be found here: *Production of NTuples*. The internal ROOT Tree has the following Branches:

**`id`** Event ID to identify correlated real sub-events.

**`nparticle`** Number of outgoing partons.

**`E/px/py/pz`** Momentum components of the partons.

**kf** Parton PDG code.

**weight** Event weight, if sub-event is treated independently.

**weight2** Event weight, if correlated sub-events are treated as single event.

**me_wgt** ME weight (w/o PDF), corresponds to 'weight'.

**me_wgt2** ME weight (w/o PDF), corresponds to 'weight2'.

**id1** PDG code of incoming parton 1.

**id2** PDG code of incoming parton 2.

**fac_scale** Factorisation scale.

**ren_scale** Renormalisation scale.

**x1** Bjorken-x of incoming parton 1.

**x2** Bjorken-x of incoming parton 2.

**x1p** x' for I-piece of incoming parton 1.

**x2p** x' for I-piece of incoming parton 2.

**nuwgt** Number of additional ME weights for loops and integrated subtraction terms.

**usr_wgt[nuwgt]** Additional ME weights for loops and integrated subtraction terms.

### 7.2.1 Computing (differential) cross sections of real correction events with statistical errors

Real correction events and their counter-events from subtraction terms are highly correlated and exhibit large cancellations. Although a treatment of sub-events as independent events leads to the correct cross section the statistical error would be greatly overestimated. In order to get a realistic statistical error sub-events belonging to the same event must be combined before added to the total cross section or a histogram bin of a differential cross section. Since in general each sub-event comes with it's own set of four momenta the following treatment becomes necessary:

1. An event here refers to a full real correction event that may contain several sub-events. All entries with the same id belong to the same event. Step 2 has to be repeated for each event.
2. Each sub-event must be checked separately whether it passes possible phase space cuts. Then for each observable add up `weight2` of all sub-events that go into the same histogram bin. These sums $x_{id}$ are the quantities to enter the actual histogram.
3. To compute statistical errors each bin must store the sum over all $x_{id}$ and the sum over all $x_{id}^2$. The cross section in the bin is given by $\langle x \rangle = \frac{1}{N} \cdot \sum x_{id}$, where $N$ is the number of events (not sub-events). The $1-\sigma$ statistical error for the bin is $\sqrt{(\langle x^2 \rangle - \langle x \rangle^2)/(N-1)}$

Note: The main difference between `weight` and `weight2` is that they refer to a different counting of events. While `weight` corresponds to each event entry (sub-event) counted separately, `weight2` counts events as defined in step 1 of the above procedure. For NLO pieces other than the real correction `weight` and `weight2` are identical.

### 7.2.2 Computation of cross sections with new PDF's

#### Born and real pieces

Notation:

```
f_a(x_a) = PDF 1 applied on parton a, F_b(x_b) = PDF 2 applied on
parton b.
```

The total cross section weight is given by:

```
weight = me_wgt f_a(x_a)F_b(x_b)
```

#### Loop piece and integrated subtraction terms

The weights here have an explicit dependence on the renormalization and factorization scales.

To take care of the renormalization scale dependence (other than via `alpha_S`) the weight `w_0` is defined as

```
w_0 = me_wgt + usr_wgts[0] log((\mu_R^new)^2/(\mu_R^old)^2) +
usr_wgts[1] 1/2 [log((\mu_R^new)^2/(\mu_R^old)^2)]^2
```

To address the factorization scale dependence the weights `w_1,...,w_8` are given by

```
w_i = usr_wgts[i+1] + usr_wgts[i+9] log((\mu_F^new)^2/(\mu_F^old)^2)
```

The full cross section weight can be calculated as

```
weight = w_0 f_a(x_a)F_b(x_b)
          + (f_a^1 w_1 + f_a^2 w_2 + f_a^3 w_3 + f_a^4 w_4) F_b(x_b)
          + (F_b^1 w_5 + F_b^2 w_6 + F_b^3 w_7 + F_b^4 w_8) f_a(x_a)
```

where

```
f_a^1 = f_a(x_a) (a=quark), \sum_q f_q(x_a) (a=gluon),
f_a^2 = f_a(x_a/x'_a)/x'_a (a=quark), \sum_q f_q(x_a/x'_a)x'_a (a=gluon),
f_a^3 = f_g(x_a),
f_a^4 = f_g(x_a/x'_a)/x'_a
```

The scale dependence coefficients `usr_wgts[0]` and `usr_wgts[1]` are normally obtained from the finite part of the virtual correction by removing renormalization terms and universal terms from dipole subtraction. This may be undesirable, especially when the loop provider splits up the calculation of the virtual correction into several pieces, like leading and sub-leading color. In this case the loop provider should control the scale dependence coefficients, which can be enforced with option `USR_WGT_MODE: false`.

**Warning:** The loop provider must support this option or the scale dependence coefficients will be invalid!

CHAPTER

EIGHT

# CUSTOMIZATION

Customizing Sherpa according to your needs.

Sherpa can be easily extended with certain user defined tools. To this extent, a corresponding C++ class must be written, and compiled into an external library:

```
$ g++ -shared \
      -I`$SHERPA_PREFIX/bin/Sherpa-config --incdir` \
      `$SHERPA_PREFIX/bin/Sherpa-config --ldflags` \
      -o libMyCustomClass.so My_Custom_Class.C
```

This library can then be loaded in Sherpa at runtime with the option `SHERPA_LDADD`, e.g.:

```
SHERPA_LDADD:
- MyCustomClass
```

Several specific examples of features which can be extended in this way are listed in the following sections.

## 8.1 Exotic physics

It is possible to add your own models to Sherpa in a straightforward way. To illustrate, a simple example has been included in the directory `Examples/Models/SM_ZPrime`, showing how to add a Z-prime boson to the Standard Model.

The important features of this example include:

- The `SM_Zprime.C` file.

  This file contains the initialisation of the Z-prime boson. The properties of the Z-prime are set here, such as mass, width, electromagnetic charge, spin etc.

- The `Interaction_Model_SM_Zprime.C` file.

  This file contains the definition of the Z-prime boson's interactions. The right- and left-handed couplings to each of the fermions are set here.

- An example `Makefile`.

  This shows how to compile the sources above into a shared library.

- The line `SHERPA_LDADD: SMZprime` in the config file.

  This line tells Sherpa to load the extra libraries created from the **.C* files above.

- The line `MODEL: SMZprime` in the config file.

  This line tells Sherpa which model to use for the run.

- The following lines in the config file:

```
PARTICLE_DATA:
  32:
    Mass: 1000
    Width: 50
```

  These lines show how you can overrule the choices you made for the properties of the new particle in the `SM_Zprime.C` file. For more information on changing parameters in Sherpa, see *Input structure* and *Parameters*.

- The lines `Zp_cpl_L: 0.3` and `Zp_cpl_R: 0.6` set the couplings to left and right handed fermions in the config file.

To use this model, create the libraries for Sherpa to use by running

```
$ make
```

in this directory. Then run Sherpa as normal:

```
$ ../../../bin/Sherpa
```

To implement your own model, copy these example files anywhere and modify them according to your needs.

Note: You don't have to modify or recompile any part of Sherpa to use your model. As long as the `SHERPA_LDADD` parameter is specified as above, Sherpa will pick up your model automatically.

Furthermore note: New physics models with an existing implementation in FeynRules, cf. [CD09] and [CdAD+11], can directly be invoked using Sherpa's support for the UFO model format, see *UFO Model Interface*.

## 8.2 Custom scale setter

You can write a custom calculator to set the factorisation, renormalisation and resummation scales. It has to be implemented as a C++ class which derives from the `Scale_Setter_Base` base class and implements only the constructor and the `Calculate` method.

Here is a snippet for a very simple one, which sets all three scales to the invariant mass of the two incoming partons.

```
#include "PHASIC++/Scales/Scale_Setter_Base.H"
#include "ATOOLS/Org/Message.H"

using namespace PHASIC;
using namespace ATOOLS;

namespace PHASIC {

  class Custom_Scale_Setter: public Scale_Setter_Base {
  protected:

  public:

    Custom_Scale_Setter(const Scale_Setter_Arguments &args) :
      Scale_Setter_Base(args)
    {
      m_scale.resize(3); // by default three scales: fac, ren, res
                         // but you can add more if you need for COUPLINGS
      SetCouplings(); // the default value of COUPLINGS is "Alpha_QCD 1", i.e.
```

(continues on next page)

(continued from previous page)

```
                    // m_scale[1] is used for running alpha_s
                    // (counting starts at zero!)
    }

    double Calculate(const std::vector<ATOOLS::Vec4D> &p,
                     const size_t &mode)
    {
      double muF=(p[0]+p[1]).Abs2();
      double muR=(p[0]+p[1]).Abs2();
      double muQ=(p[0]+p[1]).Abs2();

      m_scale[stp::fac] = muF;
      m_scale[stp::ren] = muR;
      m_scale[stp::res] = muQ;

      // Switch on debugging output for this class with:
      // Sherpa "OUTPUT=2[Custom_Scale_Setter|15]"
      DEBUG_FUNC("Calculated scales:");
      DEBUG_VAR(m_scale[stp::fac]);
      DEBUG_VAR(m_scale[stp::ren]);
      DEBUG_VAR(m_scale[stp::res]);

      return m_scale[stp::fac];
    }

  };

}

// Some plugin magic to make it available for SCALES=CUSTOM
DECLARE_GETTER(Custom_Scale_Setter,"CUSTOM",
            Scale_Setter_Base,Scale_Setter_Arguments);

Scale_Setter_Base *ATOOLS::Getter
<Scale_Setter_Base,Scale_Setter_Arguments,Custom_Scale_Setter>::
operator()(const Scale_Setter_Arguments &args) const
{
  return new Custom_Scale_Setter(args);
}

void ATOOLS::Getter<Scale_Setter_Base,Scale_Setter_Arguments,
                Custom_Scale_Setter>::
PrintInfo(std::ostream &str,const size_t width) const
{
  str<<"Custom scale scheme";
}
```

If the code is compiled into a library called libCustomScale.so, then this library is loaded dynamically at runtime with the switch `SHERPA_LDADD: CustomScale` either on the command line or in the run section, cf. *Customization*. This then allows to use the custom scale like a built-in scale setter by specifying `SCALES: CUSTOM` (cf. *SCALES*).

## 8.3 External one-loop ME

Sherpa includes only a very limited selection of one-loop matrix elements. To make full use of the implemented automated dipole subtraction it is possible to link external one-loop codes to Sherpa in order to perform full calculations at QCD next-to-leading order.

In general Sherpa can take care of any piece of the calculation except one-loop matrix elements, i.e. the born ME, the real correction, the real and integrated subtraction terms as well as the phase space integration and PDF weights for hadron collisions. Sherpa will provide sets of four-momenta and request for a specific parton level process the helicity and colour summed one-loop matrix element (more specific: the coefficients of the Laurent series in the dimensional regularization parameter epsilon up to the order epsilon^0).

An example setup for interfacing such an external one-loop code, following the Binoth Les Houches interface proposal [B+10] of the 2009 Les Houches workshop, is provided in *Zbb production*. To use the LH-OLE interface, Sherpa has to be configured with `--enable-lhole`.

The interface:

- During an initialization run Sherpa stores setup information (schemes, model information etc.) and requests a list of parton-level one-loop processes that are needed for the NLO calculation. This information is stored in a file, by default called `OLE_order.lh`. The external one-loop code (OLE) should confirm these settings/requests and write out a file `OLE_contract.lh`. Both filenames can be customised using `LHOLE_ORDERFILE: <order-file>` and `<LHOLE_CONTRACTFILE: <contract-file>`. For the syntax of these files and more details see [B+10].

  For Sherpa the output/input of the order/contract file is handled in `LH_OLE_Communicator.[CH]`. The actual interface is contained in `LH_OLE_Interface.C`. The parameters to be exchanged with the OLE are defined in the latter file via

  ```
  lhfile.AddParameter(...);
  ```

  and might require an update for specific OLE or processes. Per default, in addition to the standard options `MatrixElementSquareType`, `CorrectionType`, `IRregularisation`, `AlphasPower`, `AlphaPower` and `OperationMode` the masses and width of the W, Z and Higgs bosons and the top and bottom quarks are written out in free format, such that the respective OLE parameters can be easily synchronised.

- At runtime the communication is performed via function calls. To allow Sherpa to call the external code the functions

  ```
  void OLP_Start(const char * filename);
  void OLP_EvalSubProcess(int,double*,double,double,double*);
  ```

  which are defined and called in `LH_OLE_Interface.C` must be specified. For keywords and possible data fields passed with this functions see [B+10].

  The function `OLP_Start(...)` is called once when Sherpa is starting. The function `OLP_EvalSubProcess(...)` will be called many times for different subprocesses and momentum configurations.

The setup (cf. example *Zbb production*):

- The line `Loop_Generator: LHOLE` tells the code to use the interface for computing one-loop matrix elements.
- The switch `SHERPA_LDADD` has to be set to the appropriate library name (and path) of the one-loop generator.
- The IR regularisation scheme can be set via `LHOLE_IR_REGULARISATION`. Possible values are `DRED` (default) and `CDR`.

- Per default, Sherpa generates phase space points in the lab frame. If `LHOLE_BOOST_TO_CMS: true` is set, these phase space points are boosted to the centre of mass system before they are passed to the OLE.
- The original BLHA interface does not allow for run-time parameter passing. While this is discussed for an updated of the accord a workable solution is implemented for the use of GoSam and enabled through `LHOLE_OLP: GoSam`. The `LHOLE_BOOST_TO_CMS` is also automatically active with this setup. This, of course, can be adapted for other one-loop programs if need be.
- Sherpa's internal analysis package can be used to generate a few histograms. Thus, then when installing Sherpa the option `--enable-analysis` must be include on the command line when Sherpa is configured, see *ANALYSIS*.

## 8.4 External RNG

To use an external Random Number Generator (RNG) in Sherpa, you need to provide an interface to your RNG in an external dynamic library. This library is then loaded at runtime and Sherpa replaces the internal RNG with the one provided.

*In this case Sherpa will not attempt to set, save, read or restore the RNG*

The corresponding code for the RNG interface is

```
#include "ATOOLS/Math/Random.H"

using namespace ATOOLS;

class Example_RNG: public External_RNG {
public:
  double Get()
  {
    // your code goes here ...
  }
};// end of class Example_RNG

// this makes Example_RNG loadable in Sherpa
DECLARE_GETTER(Example_RNG,"Example_RNG",External_RNG,RNG_Key);
External_RNG *ATOOLS::Getter<External_RNG,RNG_Key,Example_RNG>::operator()(const RNG_
↪Key &) const
{ return new Example_RNG(); }
// this eventually prints a help message
void ATOOLS::Getter<External_RNG,RNG_Key,Example_RNG>::PrintInfo(std::ostream &str,
↪const size_t) const
{ str<<"example RNG interface"; }
```

If the code is compiled into a library called libExampleRNG.so, then this library is loaded dynamically in Sherpa using the command `SHERPA_LDADD: ExampleRNG` either on the command line or in `Sherpa.yaml`. If the library is bound at compile time, like e.g. in cmt, you may skip this step.

Finally Sherpa is instructed to retrieve the external RNG by specifying `EXTERNAL_RNG: Example_RNG` on the command line or in `Sherpa.yaml`.

## 8.5 External PDF

To use an external PDF (not included in LHAPDF) in Sherpa, you need to provide an interface to your PDF in an external dynamic library. This library is then loaded at runtime and it is possible within Sherpa to access all PDFs included.

The simplest C++ code to implement your interface looks as follows

```
#include "PDF/Main/PDF_Base.H"

using namespace PDF;

class Example_PDF: public PDF_Base {
public:
  void Calculate(double x,double Q2)
  {
    // calculate values x f_a(x,Q2) for all a
  }
  double GetXPDF(const ATOOLS::Flavour a)
  {
    // return x f_a(x,Q2)
  }
  virtual PDF_Base *GetCopy()
  {
    return new Example_PDF();
  }
};// end of class Example_PDF

// this makes Example_PDF loadable in Sherpa
DECLARE_PDF_GETTER(Example_PDF_Getter);
PDF_Base *Example_PDF_Getter::operator()(const Parameter_Type &args) const
{ return new Example_PDF(); }
// this eventually prints a help message
void Example_PDF_Getter::PrintInfo
(std::ostream &str,const size_t width) const
{ str<<"example PDF"; }
// this lets Sherpa initialize and unload the library
Example_PDF_Getter *p_get=NULL;
extern "C" void InitPDFLib()
{ p_get = new Example_PDF_Getter("ExamplePDF"); }
extern "C" void ExitPDFLib() { delete p_get; }
```

If the code is compiled into a library called libExamplePDFSherpa.so, then this library is loaded dynamically in Sherpa using `PDF_LIBRARY: ExamplePDFSherpa` either on the command line, or in the `Sherpa.yaml`. If the library is bound at compile time, like e.g. in cmt, you may skip this step. It is now possible to list all accessible PDF sets by specifying `SHOW_PDF_SETS: 1` on the command line.

Finally Sherpa is instructed to retrieve the external PDF by specifying `PDF_SET: ExamplePDF` on the command line or in the `Sherpa.yaml`.

## 8.6 Python Interface

Certain Sherpa classes and methods can be made available to the Python interpreter in the form of an extension module. This module can be loaded in Python and provides access to certain functionalities of the Sherpa event generator in Python. In order to build the module, Sherpa must be configured with the option `--enable-pyext`. Running `make` then invokes the automated interface generator SWIG [Bea03] to create the Sherpa module using the Python C/C++ API. SWIG version 1.3.x or later is required for a successful build. Problems might occur if more than one version of Python is present on the system since automake currently doesn't always handle multiple Python installations properly. If you have multiple Python versions installed on your system, please set the `PYTHON` environment variable to the Python 2 executable via

```
$ export PYTHON=<path-to-python2>
```

before executing the `configure` script (see. Certain Sherpa classes and methods can be made available to the Python interpreter in the form of an extension module. This module can be loaded in Python and provides access to certain functionalities of the Sherpa event generator in Python. It was designed specifically for the computation of matrix elements in python (*Using the Python interface*) and its features are currently limited to this purpose. In order to build the module, Sherpa must be configured with the option `--enable-pyext`. Running `make` then invokes the automated interface generator SWIG [Bea03] to create the Sherpa module using the Python C/C++ API. SWIG version 1.3.x or later is required for a successful build. Problems might occur if more than one version of Python is present on the system since automake currently doesn't always handle multiple Python installations properly. A possible workaround is to temporarily uninstall one version of python, configure and build Sherpa, and then reinstall the temporarily uninstalled version of Python.

The following script is a minimal example that shows how to use the Sherpa module in Python. In order to load the Sherpa module, the location where it is installed must be added to the PYTHONPATH. There are several ways to do this, in this example the sys module is used. The sys module also allows it to directly pass the command line arguments used to run the script to the initialization routine of Sherpa. The script can thus be executed using the normal command line options of Sherpa (see *Command Line Options*). Furthermore it illustrates how exceptions that Sherpa might throw can be taken care of. If a run card is present in the directory where the script is executed, the initialization of the generator causes Sherpa to compute the cross sections for the processes specified in the run card. See *Computing matrix elements for individual phase space points using the Python Interface* for an example that shows how to use the Python interface to compute matrix elements or *Generate events using scripts* to see how the interface can be used to generate events in Python.

Note that if you have compiled Sherpa with MPI support, you need to source the mpi4py module using `from mpi4py import MPI`.

```
#!/usr/bin/python
import sys
sys.path.append('<sherpa-prefix>/lib/<your-python-version>/site-packages/>')
import Sherpa

# set up the generator
Generator=Sherpa.Sherpa(len(sys.argv),sys.argv)

# initialize the generator, pass command line arguments to initialization routine
try:
  Generator.InitializeTheRun()

 # catch exceptions
 except Sherpa.Exception as exc:
   print exc
```

CHAPTER

NINE

# EXAMPLES

Some example set-ups are included in Sherpa, in the `<prefix>/share/SHERPA-MC/Examples/` directory. These may be useful to new users to practice with, or as templates for creating your own Sherpa run-cards. In this section, we will look at some of the main features of these examples.



## 9.1 Vector boson + jets production



To change any of the following LHC examples to production at different collider energies or beam types, e.g. proton anti-proton at the Tevatron, simply change the beam settings accordingly:

```
BEAMS: [2212, -2212]
BEAM_ENERGIES: 980
```

### 9.1.1 W+jets production

This is an example setup for inclusive W production at hadron colliders. The inclusive process is calculated at next-to-leading order accuracy matched to the parton shower using the MC@NLO prescription detailed in [HKSS12]. The next few higher jet multiplicities, calculated at next-to-leading order as well, are merged into the inclusive sample using the MEPS@NLO method – an extension of the CKKW method to NLO – as described in [HKSS13] and [GHK+13]. Finally, even higher multiplicities, calculated at leading order, are merged on top of that. A few more things to note are detailed below the example.

```
# set up two proton beams, each at 6.5 TeV
BEAMS: 2212
BEAM_ENERGIES: 6500

# matrix-element calculation
ME_GENERATORS:
- Comix
- Amegic
- OpenLoops

# optional: use a custom jet criterion
#SHERPA_LDADD: MyJetCriterion
#JET_CRITERION: FASTJET[A:antikt,R:0.4,y:5]

# exclude tau (15) from (massless) lepton container (90)
PARTICLE_DATA:
  15:
    Massive: 1

# pp -> W[lv]+jets
PROCESSES:
- 93 93 -> 90 91 93{4}:
    Order: {QCD: 0, EW: 2}
    CKKW: 20
    # set up MC@NLO final-state multiplicities
    2->2-4:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
      Loop_Generator: OpenLoops
    # make calculation of higher final-state multiplicities faster
    2->4-6:
      Integration_Error: 0.05

SELECTORS:
# Safety cuts to avoid PDF calls with muF < 1 GeV
- [Mass, 11, -12, 1.0, E_CMS]
- [Mass, 13, -14, 1.0, E_CMS]
- [Mass, -11, 12, 1.0, E_CMS]
- [Mass, -13, 14, 1.0, E_CMS]
```

Things to notice:

- The `Order` in the process definition in a multi-jet merged setup defines the order of the core process (here `93 93 -> 90 91` with two electroweak couplings). The additional strong couplings for multi-jet production are implicitly understood.
- The settings necessary for NLO accuracy are restricted to the 2->2,3,4 processes using the `2->2-4` key below the `# set up MC@NLO ...` comment. The example can be converted into a simple MENLOPS setup by using `2->2` instead, or into an MEPS setup by removing these lines altogether. Thus one can study the effect of incorporating higher-order matrix elements.
- The number of additional LO jets can be varied through changing the integer within the curly braces in the `Process` definition, which gives the maximum number of additional partons in the matrix elements.
- `OpenLoops` is used here as the provider of the one-loop matrix elements for the respective multiplicities.
- Tau leptons are set massive in order to exclude them from the massless lepton container (`90`).
- As both Comix and Amegic are specified as matrix element generators to be used, Amegic has to be specified to be used for all MC@NLO multiplicities using `ME_Generator: Amegic`. Additionally, we specify `RS_ME_Generator: Comix` such that the subtracted real-emission bit of the NLO matrix elements is calculated more efficiently with Comix instead of Amegic. This combination is currently the only one supported for NLO-matched/merged setups.

The jet criterion used to define the matrix element multiplicity in the context of multijet merging can be supplied by the user. As an example the source code file `./Examples/V_plus_Jets/LHC_WJets/My_JetCriterion.C` provides such an alternative jet criterion. It can be compiled using `SCons` via executing `scons` in that directory (edit the `SConstruct` file accordingly). The newly created library is linked at run time using the `SHERPA_LDADD` flag. The new jet criterion is then evoked by `JET_CRITERION`.

### 9.1.2 Z+jets production

This is an example setup for inclusive Z production at hadron colliders. The inclusive process is calculated at next-to-leading order accuracy matched to the parton shower using the MC@NLO prescription detailed in [HKSS12]. The next few higher jet multiplicities, calculated at next-to-leading order as well, are merged into the inclusive sample using the MEPS@NLO method – an extension of the CKKW method to NLO – as described in [HKSS13] and [GHK+13]. Finally, even higher multiplicities, calculated at leading order, are merged on top of that. A few things to note are detailed below the previous *W+jets production* example and apply also to this example.

```
# Sherpa configuration for Z+Jets production

# set up beams for LHC run 2
BEAMS: 2212
BEAM_ENERGIES: 6500

# matrix-element calculation
ME_GENERATORS:
- Comix
- Amegic
- OpenLoops

# pp -> Z[ee]+jets
PROCESSES:
- 93 93 -> 11 -11 93{1}:
    Order: {QCD: 0, EW: 2}
    CKKW: 20
    # set up NLO+PS final-state multiplicities
    2->2:
      NLO_Mode: MC@NLO
```

(continues on next page)

(continued from previous page)

```
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
      Loop_Generator: OpenLoops
    # make integration of higher final-state multiplicities faster
    2->3:
      Integration_Error: 0.05

SELECTORS:
- [Mass, 11, -11, 66, E_CMS]
```

### 9.1.3 W+bb production

This example is currently broken. Please contact the *Authors* for more information.

### 9.1.4 Zbb production

```
BEAMS: 2212
BEAM_ENERGIES: 6500

# general settings
EVENTS: 1M

# me generator settings
ME_GENERATORS: [Comix, Amegic, LHOLE]

HARD_DECAYS:
  Enabled: true
  Mass_Smearing: 0
  Channels:
    23 -> 11 -11: {Status: 2}
    23 -> 13 -13: {Status: 2}

PARTICLE_DATA:
  5:
    Massive: true
    Mass: 4.75  # consistent with MSTW 2008 nf 4 set
  23:
    Width: 0
    Stable: 0

MI_HANDLER: None
FRAGMENTATION: None
CORE_SCALE: VAR{H_T2+sqr(91.188)}
PDF_LIBRARY: MSTW08Sherpa
PDF_SET: mstw2008nlo_nf4

PROCESSES:
- 93 93 -> 23 5 -5:
    NLO_Mode: MC@NLO
    NLO_Order: {QCD: 1, EW: 0}
    ME_Generator: Amegic
    RS_ME_Generator: Comix
```

(continues on next page)

(continued from previous page)

```
    Loop_Generator: LHOLE
    Order: {QCD: Any, EW: 1}

SELECTORS:
- FastjetFinder:
    Algorithm: antikt
    N: 2
    PTMin: 5.0
    DR: 0.5
    EtaMax: 5
    Nb: 2
```

Things to notice:

- The matrix elements are interfaced via the Binoth Les Houches interface proposal [B+10], [A+b], *External one-loop ME*.
- The Z-boson is stable in the hard matrix elements. It is decayed using the internal decay module, indicated by the settings `HARD_DECAYS:Enabled: true` and `PARTICLE_DATA:23:Stable: 0`.
- fjcore from FastJet is used to regularize the hard cross section. We require two b-jets, indicated by `Nb: 2` at the end of the `FastjetFinder` options.
- Four-flavour PDF are used to comply with the calculational setup.

## 9.2 Jet production



### 9.2.1 Jet production



To change any of the following LHC examples to production at the Tevatron simply change the beam settings to

```
BEAMS: [2212, -2212]
BEAM_ENERGIES: 980
```

### MC@NLO setup for dijet and inclusive jet production

This is an example setup for dijet and inclusive jet production at hadron colliders at next-to-leading order precision matched to the parton shower using the MC@NLO prescription detailed in [HKSS12] and [HS12]. A few things to note are detailed below the example.

```
# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500.0

SCALES: FASTJET[A:antikt,PT:20.0,ET:0,R:0.4,M:0]{0.0625*H_T2}{0.0625*H_T2}{0.
↪25*PPerp2(p[3])}
ME_GENERATORS: [Amegic, OpenLoops]

PROCESSES:
- 93 93 -> 93 93:
    NLO_Mode: MC@NLO
    NLO_Order: {QCD: 1, EW: 0}
    Loop_Generator: OpenLoops
    Order: {QCD: 2, EW: 0}

SELECTORS:
- FastjetFinder:
    Algorithm: antikt
    N: 2
    PTMin: 10.0
    ETMin: 0.0
    DR: 0.4
- FastjetFinder:
    Algorithm: antikt
    N: 1
    PTMin: 20.0
    ETMin: 0.0
    DR: 0.4
```

Things to notice:

- Asymmetric cuts are implemented (relevant to the RS-piece of an MC@NLO calculation) by requiring at least two jets with pT > 10 GeV, one of which has to have pT > 20 GeV.
- Both the factorisation and renormalisation scales are set to the above defined scale factors times a quarter of the scalar sum of the transverse momenta of all anti-kt jets (R = 0.4, pT > 20 GeV) found on the ME-level before any parton shower emission. See *SCALES* for details on scale setters.
- The resummation scale, which sets the maximum scale of the additional emission to be resummed by the parton shower, is set to the above defined resummation scale factor times half of the transverse momentum of the softer of the two jets present at Born level.
- The external generator OpenLoops provides the one-loop matrix elements.
- The `NLO_Mode` is set to `MC@NLO`.

#### MEPS setup for jet production

```
BEAMS: 2212
BEAM_ENERGIES: 6500

PROCESSES:
- 93 93 -> 93 93 93{0}:
    Order: {QCD: Any, EW: 0}
    CKKW: 20
    Integration_Error: 0.02

SELECTORS:
- NJetFinder:
    N: 2
    PTMin: 20.0
    ETMin: 0.0
    R: 0.4
    Exp: -1
```

Things to notice:

- `Order` is set to `{QCD: Any, EW: 0}`. This ensures that all final state jets are produced via the strong interaction.
- An `NJetFinder` selector is used to set a resolution criterion for the two jets of the core process. This is necessary because the "CKKW" tag does not apply any cuts to the core process, but only to the extra-jet matrix elements, see *Multijet merged event generation with Sherpa*.

### 9.2.2 Jets at lepton colliders



This section contains two setups to describe jet production at LEP I, either through multijet merging at leading order accuracy or at next-to-leading order accuracy.

#### MEPS setup for ee->jets

This example shows a LEP set-up, with electrons and positrons colliding at a centre of mass energy of 91.2 GeV.

```
BEAMS: [11, -11]
BEAM_ENERGIES: 45.6

ALPHAS(MZ): 0.1188
ORDER_ALPHAS: 1

PROCESSES:
- 11 -11 -> 93 93 93{3}:
    CKKW: pow(10,-2.25/2.00)*E_CMS
    Order: {QCD: Any, EW: 2}
```

Things to notice:

- The running of alpha_s is set to leading order and the value of alpha_s at the Z-mass is set.
- Note that initial-state radiation is enabled by default. See *ISR parameters* on how to disable it if you want to evaluate the (unphysical) case where the energy for the incoming leptons is fixed.

### MEPS@NLO setup for ee->jets

This example expands upon the above setup, elevating its description of hard jet production to next-to-leading order.

```
# collider setup
BEAMS: [11, -11]
BEAM_ENERGIES: 45.6

TAGS:
  # tags for process setup
  YCUT: 2.0
  # tags for ME generators
  LOOPGEN0: Internal
  LOOPGEN1: <my-loop-gen-for-3j>
  LOOPGEN2: <my-loop-gen-for-4j>
  LOOPMGEN: <my=loop-gen-for-massive-2j>

# settings for ME generators
ME_GENERATORS:
  - Comix
  - Amegic
  - $(LOOPGEN0)
  - $(LOOPGEN1)
  - $(LOOPGEN2)
  - $(LOOPMGEN)
AMEGIC: {INTEGRATOR: 4}

# model parameters
MODEL: SM
ALPHAS(MZ): 0.118
PARTICLE_DATA: {5: {Massive: true}}

PROCESSES:
- 11 -11 -> 93 93 93{3}:
    CKKW: pow(10,-$(YCUT)/2.00)*E_CMS
    Order: {QCD: Any, EW: 2}
    RS_Enhance_Factor: 10
    2->2: { Loop_Generator: $(LOOPGEN0) }
    2->3: { Loop_Generator: $(LOOPGEN1) }
    2->4: { Loop_Generator: $(LOOPGEN2) }
    2->2-4:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
- 11 -11 -> 5 -5 93{3}:
    CKKW: pow(10,-$(YCUT)/2.00)*E_CMS
    Order: {QCD: Any, EW: 2}
    Loop_Generator: $(LOOPMGEN)
    RS_Enhance_Factor: 10
    2->2:
      NLO_Mode: MC@NLO
```

(continues on next page)

(continued from previous page)

```
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
- 11 -11 -> 5 5 -5 -5 93{1}:
    CKKW: pow(10,-$(YCUT)/2.00)*E_CMS
    Order: {QCD: Any, EW: 2}
    Cut_Core: 1
```

Things to notice:

- the b-quark mass has been enabled for the matrix element calculation (the default is massless) because it is not negligible for LEP energies
- the b b-bar and b b b-bar b-bar processes are specified separately because the `93` particle container contains only partons set massless in the matrix element calculation, see *Particle containers*.
- model parameters can be modified in the config file; in this example, the value of alpha_s at the Z mass is set.

## 9.3 Higgs boson + jets production



### 9.3.1 H production in gluon fusion with interference effects

This is a setup for inclusive Higgs production through gluon fusion at hadron colliders. The inclusive process is calculated at next-to-leading order accuracy, including all interference effects between Higgs-boson production and the SM gg->yy background. The corresponding matrix elements are taken from [BDS02] and [DL].

```
# collider parameters
BEAMS: 2212
BEAM_ENERGIES: 6500

# generator parameters
EVENTS: 1M
EVENT_GENERATION_MODE: Weighted
AMEGIC: {ALLOW_MAPPING: 0}
ME_GENERATORS: [Amegic, Higgs]
SCALES: VAR{Abs2(p[2]+p[3])}

# physics parameters
```

(continues on next page)

(continued from previous page)

```
PARTICLE_DATA:
  4:  {Yukawa: 1.42}
  5:  {Yukawa: 4.92}
  15: {Yukawa: 1.777}
EW_SCHEME: 3
RUN_MASS_BELOW_POLE: 1

PROCESSES:
- 93 93 -> 22 22:
    NLO_Mode: Fixed_Order
    Order: {QCD: Any, EW: 2}
    NLO_Order: {QCD: 1, EW: 0}
    Enable_MHV: 12
    Loop_Generator: Higgs
    Integrator: PS2
    RS_Integrator: PS3

SELECTORS:
- HiggsFinder:
    PT1: 40
    PT2: 30
    Eta: 2.5
    MassRange: [100, 150]
- [IsolationCut, 22, 0.4, 2, 0.025]
```

Things to notice:

- This calculation is at fixed-order NLO.
- All scales, i.e. the factorisation, renormalisation and resummation scales are set to the invariant mass of the di-photon pair.
- Dedicated phase space generators are used by setting `Integrator: PS2` and `RS_Integrator: PS3`, cf. *Integrator*.

To compute the interference contribution only, as was done in [DL], one can set `HIGGS_INTERFERENCE_ONLY: 1`. By default, all partonic processes are included in this simulation, however, it is sensible to disable quark initial states at the leading order. This is achieved by setting `HIGGS_INTERFERENCE_MODE: 3`.

One can also simulate the production of a spin-2 massive graviton in Sherpa using the same input card by setting `HIGGS_INTERFERENCE_SPIN: 2`. Only the massive graviton case is implemented, specifically the scenario where k_q=k_g. NLO corrections are approximated, as the gg->X->yy and qq->X->yy loop amplitudes have not been computed so far.

### 9.3.2 H+jets production in gluon fusion

This is an example setup for inclusive Higgs production through gluon fusion at hadron colliders used in [HKS]. The inclusive process is calculated at next-to-leading order accuracy matched to the parton shower using the MC@NLO prescription detailed in [HKSS12]. The next few higher jet multiplicities, calculated at next-to-leading order as well, are merged into the inclusive sample using the MEPS@NLO method – an extension of the CKKW method to NLO – as described in [HKSS13] and [GHK+13]. Finally, even higher multiplicities, calculated at leading order, are merged on top of that. A few things to note are detailed below the example.

```
# collider parameters
BEAMS: 2212
BEAM_ENERGIES: 6500
```

(continues on next page)

(continued from previous page)

```
# settings for ME generators
ME_GENERATORS: [Comix, Amegic, Internal, MCFM]

# settings for hard decays
HARD_DECAYS:
  Enabled: true
  Channels:
    25 -> 22 22: {Status: 2}
  Apply_Branching_Ratios: false

# model parameters
MODEL: HEFT
PARTICLE_DATA:
  25: {Mass: 125, Width: 0}

PROCESSES:
- 93 93 -> 25 93{2}:
    Order: {QCD: Any, EW: 0, HEFT: 1}
    CKKW: 30
    2->1-2: { Loop_Generator: Internal }
    2->3:   { Loop_Generator: MCFM }
    2->1-3:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
```

Things to notice:

- The example can be converted into a simple MENLOPS setup by replacing `2->1-3` with `2->1`, or into an MEPS setup with `2->0`, to study the effect of incorporating higher-order matrix elements.
- Providers of the one-loop matrix elements for the respective multiplicities are set using `Loop_Generator`. For the two simplest cases Sherpa can provide it internally. Additionally, MCFM is interfaced for the H+2jet process, cf. *MCFM interface*.
- To enable the Higgs to decay to a pair of photons, for example, the hard decays are invoked. For details on the hard decay handling and how to enable specific decay modes see *Hard decays*.

### 9.3.3 H+jets production in gluon fusion with finite top mass effects

This is example is similar to *H+jets production in gluon fusion* but with finite top quark mass taken into account as described in [B+15] for all merged jet multiplicities. Mass effects in the virtual corrections are treated in an approximate way. In case of the tree-level contributions, including real emission corrections, no approximations are made concerning the mass effects.

```
# collider parameters
BEAMS: 2212
BEAM_ENERGIES: 6500

# settings for ME generators
ME_GENERATORS: [Amegic, Internal, OpenLoops]

# settings for hard decays
HARD_DECAYS:
```

(continues on next page)

(continued from previous page)

```
  Enabled: true
  Channels:
    25 -> 22 22: {Status: 2}
  Apply_Branching_Ratios: false

# model parameters
MODEL: HEFT
PARTICLE_DATA:
  25: {Mass: 125, Width: 0}

# finite top mass effects
KFACTOR: GGH
OL_IGNORE_MODEL: true
OL_PARAMETERS:
  preset: 2
  allowed_libs: pph2,pphj2,pphjj2
  psp_tolerance: 1.0e-7

PROCESSES:
- 93 93 -> 25 93{1}:
    Order: {QCD: 0, EW: 0, HEFT: 1}
    CKKW: 30
    Loop_Generator: Internal
    2->1-2:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
```

Things to notice:

- One-loop matrix elements from OpenLoops [CMP12] are used in order to correct for top mass effects. Sherpa must therefore be compiled with OpenLoops support to run this example. Also, the OpenLoops process libraries listed in the run card must be installed.
- The maximum jet multiplicities that can be merged in this setup are limited by the availability of loop matrix elements used to correct for finite top mass effects.
- The comments in *H+jets production in gluon fusion* apply here as well.

### 9.3.4 H+jets production in associated production

This section collects example setups for Higgs boson production in association with vector bosons

- *Higgs production in association with W bosons and jets*
- *Higgs production in association with Z bosons and jets*
- *Higgs production in association with lepton pairs*

### Higgs production in association with W bosons and jets

This is an example setup for Higgs boson production in association with a W boson and jets, as used in [HKP+]. It uses the MEPS@NLO method to merge pp->WH and pp->WHj at next-to-leading order accuracy and adds pp->WHjj at leading order. The Higgs boson is decayed to W-pairs and all W decay channels resulting in electrons or muons are accounted for, including those with intermediate taus.

```
# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

ME_GENERATORS: [Comix, Amegic, OpenLoops]

# define custom particle container for easy process declaration
PARTICLE_CONTAINERS:
  900: {Name: W, Flavours: [24, -24]}
  901: {Name: lightflavs, Flavours: [1, -1, 2, -2, 3, -3, 4, -4, 21]}
NLO_CSS_DISALLOW_FLAVOUR: 5

# particle properties (ME widths need to be zero if external)
PARTICLE_DATA:
  24: {Width: 0}
  25: {Mass: 125.5, Width: 0}
  15: {Stable: 0, Massive: true}


# hard decays setup, specify allowed decay channels, ie.:
# h->Wenu, h->Wmunu, h->Wtaunu, W->enu, W->munu, W->taunu, tau->enunu, tau->munununu +␣
↪cc
HARD_DECAYS:
  Enabled: true
  Channels:
    25 -> 24 -12 11: {Status: 2}
    25 -> 24 -14 13: {Status: 2}
    25 -> 24 -16 15: {Status: 2}
    25 -> -24 12 -11: {Status: 2}
    25 -> -24 14 -13: {Status: 2}
    25 -> -24 16 -15: {Status: 2}
    24 -> 12 -11: {Status: 2}
    24 -> 14 -13: {Status: 2}
    24 -> 16 -15: {Status: 2}
    -24 -> -12 11: {Status: 2}
    -24 -> -14 13: {Status: 2}
    -24 -> -16 15: {Status: 2}
    15 -> 16 -12 11: {Status: 2}
    15 -> 16 -14 13: {Status: 2}
    -15 -> -16 12 -11: {Status: 2}
    -15 -> -16 14 -13: {Status: 2}
  Decay_Tau: 1
  Apply_Branching_Ratios: 0

PROCESSES:
- 901 901 -> 900 25 901{2}:
    Order: {QCD: 0, EW: 2}
    CKKW: 30
    2->2-3:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
```

(continues on next page)

(continued from previous page)

```
    ME_Generator: Amegic
    RS_ME_Generator: Comix
    Loop_Generator: OpenLoops
```

Things to notice:

- Two custom particle container, cf. *Particle containers*, have been declared, facilitating the process declaration.
- As the bottom quark is treated as being massless by default, a five flavour calculation is performed. The particle container ensures that no external bottom quarks, however, are considered to resolve the overlap with single top and top pair processes.
- OpenLoops [CMP12] is used as the provider of the one-loop matrix elements.
- To enable the decays of the Higgs, W boson and tau lepton the hard decay handler is invoked. For details on the hard decay handling and how to enable specific decay modes see *Hard decays*.

### Higgs production in association with Z bosons and jets

This is an example setup for Higgs boson production in association with a Z boson and jets, as used in [HKP+]. It uses the MEPS@NLO method to merge pp->ZH and pp->ZHj at next-to-leading order accuracy and adds pp->ZHjj at leading order. The Higgs boson is decayed to W-pairs. All W and Z bosons are allowed to decay into electrons, muons or tau leptons. The tau leptons are then allowed to decay into all possible partonic channels, leptonic and hadronic, to allow for all possible trilepton signatures, unavoidably producing two and four lepton events as well.

```
# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

ME_GENERATORS: [Comix, Amegic, OpenLoops]

# define custom particle container for easy process declaration
PARTICLE_CONTAINERS:
  901: {Name: lightflavs, Flavours: [1, -1, 2, -2, 3, -3, 4, -4, 21]}
NLO_CSS_DISALLOW_FLAVOUR: 5

# particle properties (ME widths need to be zero if external)
PARTICLE_DATA:
  23: {Width: 0}
  25: {Mass: 125.5, Width: 0}
  15: {Stable: 0, Massive: true}


# hard decays setup, specify allowed decay channels
# h->Wenu, h->Wmunu, h->Wtaunu, W->enu, W->munu, W->taunu,
# Z->ee, Z->mumu, Z->tautau, tau->any + cc
HARD_DECAYS:
  Enabled: true
  Channels:
    25 -> 24 -12 11: {Status: 2}
    25 -> 24 -14 13: {Status: 2}
    25 -> 24 -16 15: {Status: 2}
    25 -> -24 12 -11: {Status: 2}
    25 -> -24 14 -13: {Status: 2}
    25 -> -24 16 -15: {Status: 2}
    24 -> 12 -11: {Status: 2}
```

(continues on next page)

(continued from previous page)

```
    24 -> 14 -13: {Status: 2}
    24 -> 16 -15: {Status: 2}
    23 -> 15 -15: {Status: 2}
    -24 -> -12 11: {Status: 2}
    -24 -> -14 13: {Status: 2}
    -24 -> -16 15: {Status: 2}
    15 -> 16 -12 11: {Status: 2}
    15 -> 16 -14 13: {Status: 2}
    -15 -> -16 12 -11: {Status: 2}
    -15 -> -16 14 -13: {Status: 2}
    15 -> 16 -2 1: {Status: 2}
    15 -> 16 -4 3: {Status: 2}
    -15 -> -16 2 -1: {Status: 2}
    -15 -> -16 4 -3: {Status: 2}
  Decay_Tau: 1
  Apply_Branching_Ratios: 0

PROCESSES:
- 901 901 -> 23 25 901{2}:
    Order: {QCD: 0, EW: 2}
    CKKW: 30
    2->2-3:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
      Loop_Generator: OpenLoops
```

Things to notice:

- A custom particle container, cf. *Particle containers*, has been declared, facilitating the process declaration.
- As the bottom quark is treated as being massless by default, a five flavour calculation is performed. The particle container ensures that no external bottom quarks, however, are considered to resolve the overlap with single top and top pair processes.
- OpenLoops [CMP12] is used as the provider of the one-loop matrix elements.
- To enable the decays of the Higgs, W and Z bosons and tau lepton the hard decay handler is invoked. For details on the hard decay handling and how to enable specific decay modes see *Hard decays*.

### Higgs production in association with lepton pairs

This is an example setup for Higgs boson production in association with an electron-positron pair using the MC@NLO technique. The Higgs boson is decayed to b-quark pairs. Contrary to the previous examples this setup does not use on-shell intermediate vector bosons in its matrix element calculation.

```
BEAMS: 2212
BEAM_ENERGIES: 6500

ME_GENERATORS: [Comix, Amegic, OpenLoops]

CORE_SCALE: VAR{Abs2(p[2]+p[3]+p[4])}

PARTICLE_DATA:
  5: {Massive: true}
```

(continues on next page)

(continued from previous page)

```
  15: {Massive: true}
  25: {Stable: 0, Width: 0.0}

# hard decays setup, specify allowed decay channels h->bb
HARD_DECAYS:
  Enabled: true
  Channels:
    25 -> 5 -5: {Status: 2}
  Apply_Branching_Ratios: false

PROCESSES:
- 93 93 -> 11 -11 25:
    Order: {QCD: 0, EW: 3}
    NLO_Mode: MC@NLO
    NLO_Order: {QCD: 1, EW: 0}
    Loop_Generator: OpenLoops
    ME_Generator: Amegic
    RS_ME_Generator: Comix
    Integration_Error: 0.1
```

Things to notice:

- The central scale is set to the invariant mass of the Higgs boson and the lepton pair.
- As the bottom quark is set to be treated massively, a four flavour calculation is performed.
- OpenLoops [CMP12] is used as the provider of the one-loop matrix elements.
- To enable the decays of the Higgs the hard decay handler is invoked. For details on the hard decay handling and how to enable specific decay modes see *Hard decays*.

### 9.3.5 Associated t anti-t H production at the LHC

This set-up illustrates the interface to an external loop matrix element generator as well as the possibility of specifying hard decays for particles emerging from the hard interaction. The process generated is the production of a Higgs boson in association with a top quark pair from two light partons in the initial state. Each top quark decays into an (anti-)bottom quark and a W boson. The W bosons in turn decay to either quarks or leptons.

```
BEAMS: 2212
BEAM_ENERGIES: 6500

ME_GENERATORS: [Comix, Amegic, OpenLoops]
# to use a dedicated loop library you can run `scons` in this Example directory and␣
↪replace OpenLoops with TTH throughout this run card

CORE_SCALE: VAR{sqr(175+125/2)}

PARTICLE_DATA:
  5: {Yukawa: 4.92}
  6: {Stable: 0, Width: 0.0}
  24: {Stable: 0}
  25: {Stable: 0, Width: 0.0}

# hard decays setup, specify allowed decay channels h->bb
HARD_DECAYS:
  Enabled: true
```

(continues on next page)

(continued from previous page)

```
  Channels:
    25 -> 5 -5: {Status: 2}
  Apply_Branching_Ratios: false

PROCESSES:
- 93 93 -> 25 6 -6:
    Order: {QCD: 2, EW: 1}
    NLO_Mode: MC@NLO
    NLO_Order: {QCD: 1, EW: 0}
    Loop_Generator: OpenLoops
    ME_Generator: Amegic
    RS_ME_Generator: Comix
    Integration_Error: 0.1
```

Things to notice:

- The virtual matrix elements is interfaced from OpenLoops.
- The top quarks are stable in the hard matrix elements. They are decayed using the internal decay module, indicated by the settings in the `HARD_DECAYS` and `PARTICLE_DATA` blocks.
- Widths of top and Higgs are set to 0 for the matrix element calculation. A kinematical Breit-Wigner distribution will be imposed a-posteriori in the decay module.
- The Yukawa coupling of the b-quark has been set to a non-zero value to allow the H->bb decay channel even despite keeping the b-quark massless for a five-flavour-scheme calculation.
- Higgs BRs are not included in the cross section (`Apply_Branching_Ratios: false`) as they will be LO only and not include loop-induced decays

## 9.4 Top quark (pair) + jets production



### 9.4.1 Top quark pair production

```
# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# scales
EXCLUSIVE_CLUSTER_MODE: 1
CORE_SCALE: TTBar

# me generator settings
ME_GENERATORS: [Comix, Amegic, OpenLoops]

# decays
HARD_DECAYS:
  Enabled: true
```

(continues on next page)

(continued from previous page)

```
  Channels:
    24 -> 2 -1: {Status: 0}
    24 -> 4 -3: {Status: 0}
    -24 -> -2 1: {Status: 0}
    -24 -> -4 3: {Status: 0}

# particle properties (width of external particles of the MEs must be zero)
PARTICLE_DATA:
  6: {Width: 0}

PROCESSES:
- 93 93 -> 6 -6 93{3}:
    Order: {QCD: 2, EW: 0}
    CKKW: 20
    2->2-3:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
      Loop_Generator: OpenLoops
    2->5-8:
      Max_N_Quarks: 6
      Integration_Error: 0.05
```

Things to notice:

- We use OpenLoops to compute the virtual corrections [CMP12].
- We match matrix elements and parton showers using the MC@NLO technique for massive particles, as described in [HHL+13].
- A non-default METS core scale setter is used, cf. *METS scale setting with multiparton core processes*
- We enable top decays through the internal decay module using `HARD_DECAYS:Enabled: true`

### 9.4.2 Production of a top quark pair in association with a W-boson

```
# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# me generator settings
ME_GENERATORS: [Comix, Amegic, OpenLoops]

# settings for hard decays
HARD_DECAYS:
  Enabled: true
  Channels:
    24 -> 2 -1: {Status: 0}
    24 -> 4 -3: {Status: 0}
    24 -> 16 -15: {Status: 0}

# model parameters
PARTICLE_DATA:
  6: {Width: 0}
  24: {Width: 0}
```

(continues on next page)

(continued from previous page)

```
# technical parameters
EXCLUSIVE_CLUSTER_MODE: 1

PROCESSES:
- 93 93 -> 6 -6 24:
    NLO_Mode: MC@NLO
    NLO_Order: {QCD: 1, EW: 0}
    Order: {QCD: 2, EW: 1}
    ME_Generator: Amegic
    RS_ME_Generator: Comix
    Loop_Generator: OpenLoops
```

Things to notice:

- Hard decays are enabled through `HARD_DECAYS:Enabled: true`.
- Top quarks and W bosons are final states in the hard matrix elements, so their widths are set to zero using `Width: 0` in their `PARTICLE_DATA` settings.
- Certain decay channels are disabled using `Status: 0` in the `Channels` sub-settings of the `HARD_DECAYS` setting.

## 9.5 Single-top production in the s, t and tW channel

In this section, examples for single-top production in three different channels are described. For the channel definitions and a validation of these setups, see [BSK].



### 9.5.1 t-channel single-top production

```
# SHERPA run card for t-channel single top-quark production at MC@NLO
# and N_f = 5

# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# me generator settings
ME_GENERATORS: [Comix, Amegic, OpenLoops]

# scales
# CORESCALE SingleTop:
#   use Mandelstam \hat{t} for s-channel 2->2 core process
CORE_SCALE: SingleTop
```

(continues on next page)

(continued from previous page)

```
# disable hadronic W decays
HARD_DECAYS:
  Enabled: true
  Channels:
    24 -> 2 -1: {Status: 0}
    24 -> 4 -3: {Status: 0}
    -24 -> -2 1: {Status: 0}
    -24 -> -4 3: {Status: 0}

# choose EW Gmu input scheme
EW_SCHEME: 3

# required for using top-quark in ME
PARTICLE_DATA: { 6: {Width: 0} }

PROCESSES:
- 93 93 -> 6 93:
    NLO_Mode: MC@NLO
    NLO_Order: {QCD: 1, EW: 0}
    Order: {QCD: 0, EW: 2}
    ME_Generator: Amegic
    RS_ME_Generator: Comix
    Loop_Generator: OpenLoops
    Min_N_TChannels: 1   # require t-channel W
```

Things to notice:

- We use OpenLoops to compute the virtual corrections [CMP12].
- We match matrix elements and parton showers using the MC@NLO technique for massive particles, as described in [HHL+13].
- A non-default METS core scale setter is used, cf. *METS scale setting with multiparton core processes*
- We enable top and W decays through the internal decay module using `HARD_DECAYS:Enabled: true`. The W is restricted to its leptonic decay channels.
- By setting `Min_N_TChannels: 1`, only t-channel diagrams are used for the calculation

### 9.5.2 t-channel single-top production with N_f=4

```
# SHERPA run card for t-channel single top-quark production at MC@NLO
# and N_f = 4

# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# me generator settings
ME_GENERATORS: [Comix, Amegic, OpenLoops]

# scales
#   muR = transverse momentum of the bottom
#   muF = muQ = transverse momentum of the top
CORE_SCALE: VAR{MPerp2(p[2])}{MPerp2(p[3])}{MPerp2(p[2])}
```

(continues on next page)

(continued from previous page)

```
# disable hadronic W decays
HARD_DECAYS:
  Enabled: true
  Channels:
    24 -> 2 -1: {Status: 0}
    24 -> 4 -3: {Status: 0}
    -24 -> -2 1: {Status: 0}
    -24 -> -4 3: {Status: 0}

# choose EW Gmu input scheme
EW_SCHEME: 3

PARTICLE_DATA:
  6: {Width: 0}  # required for using top-quark in ME
  5: {Massive: true, Mass: 4.18}  # mass as in NNPDF30_nlo_as_0118_nf_4

# configure for N_f = 4
PDF_LIBRARY: LHAPDFSherpa
PDF_SET: NNPDF30_nlo_as_0118_nf_4
ALPHAS: {USE_PDF: 1}

PROCESSES:
- 93 93 -> 6 -5 93:
    NLO_Mode: MC@NLO
    NLO_Order: {QCD: 1, EW: 0}
    Order: {QCD: 1, EW: 2}
    ME_Generator: Amegic
    RS_ME_Generator: Comix
    Loop_Generator: OpenLoops
    Min_N_TChannels: 1  # require t-channel W
```

Things to notice:

- We use an Nf4 PDF and use its definition of the bottom mass
- See *t-channel single-top production* for more comments

### 9.5.3 s-channel single-top production

```
# SHERPA run card for s-channel single top-quark production at MC@NLO
# and N_f = 5

# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# me generator settings
ME_GENERATORS: [Comix, Amegic, OpenLoops]

# scales
# CORESCALE SingleTop:
#   use Mandelstam \hat{s} for s-channel 2->2 core process
CORE_SCALE: SingleTop

# disable hadronic W decays
HARD_DECAYS:
```

(continues on next page)

(continued from previous page)

```
  Enabled: true
  Channels:
    24 -> 2 -1: {Status: 0}
    24 -> 4 -3: {Status: 0}
    -24 -> -2 1: {Status: 0}
    -24 -> -4 3: {Status: 0}

# choose EW Gmu input scheme
EW_SCHEME: 3

# required for using top-quark in ME
PARTICLE_DATA: { 6: {Width: 0} }

# there is no bottom in the initial-state in s-channel production
PARTICLE_CONTAINERS:
  900: {Name: lj, Flavours: [1, -1, 2, -2, 3, -3, 4, -4, 21]}

PROCESSES:
- 900 900 -> 6 93:
    NLO_Mode: MC@NLO
    NLO_Order: {QCD: 1, EW: 0}
    Order: {QCD: 0, EW: 2}
    ME_Generator: Amegic
    RS_ME_Generator: Comix
    Loop_Generator: OpenLoops
    Max_N_TChannels: 0  # require s-channel W
```

Things to notice:

- By excluding the bottom quark from the initial-state at Born level using `PARTICLE_CONTAINER`, and by setting `Max_N_TChannels: 0`, only s-channel diagrams are used for the calculation
- See *t-channel single-top production* for more comments

### 9.5.4 tW-channel single-top production

```
# SHERPA run card for tW-channel single top-quark production at MC@NLO
# and N_f = 5

# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# me generator settings
ME_GENERATORS: [Comix, Amegic, OpenLoops]

# scales
# mu = transverse momentum of the top
CORE_SCALE: VAR{MPerp2(p[3])}{MPerp2(p[3])}{MPerp2(p[3])}

# disable hadronic W decays
HARD_DECAYS:
  Enabled: true
  Channels:
    24 -> 2 -1: {Status: 0}
    24 -> 4 -3: {Status: 0}
```

(continues on next page)

(continued from previous page)

```
    -24 -> -2 1: {Status: 0}
    -24 -> -4 3: {Status: 0}

# choose EW Gmu input scheme
EW_SCHEME: 3

# required for using top-quark/W-boson in ME
PARTICLE_DATA:
  6: {Width: 0}
  24: {Width: 0}

PROCESSES:
- 93 93 -> 6 -24:
    No_Decay: -6  # remove ttbar diagrams
    NLO_Mode: MC@NLO
    NLO_Order: {QCD: 1, EW: 0}
    Order: {QCD: 1, EW: 1}
    ME_Generator: Amegic
    RS_ME_Generator: Comix
    Loop_Generator: OpenLoops
```

Things to notice:

- By setting `No_Decay: -6`, the doubly-resonant TTbar diagrams are removed. Only the singly-resonant diagrams remain as required by the definition of the channel.
- See *t-channel single-top production* for more comments

## 9.6 Vector boson pairs + jets production

- *Dilepton, missing energy and jets production*
- *Dilepton, missing energy and jets production (gluon initiated)*
- *Four lepton and jets production*
- *Four lepton and jets production (gluon initiated)*
- *WZ production with jets production*
- *Same sign dilepton, missing energy and jets production*

### 9.6.1 Dilepton, missing energy and jets production

```
# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# me generator settings
ME_GENERATORS: [Comix, Amegic, OpenLoops]
METS: { CLUSTER_MODE: 16 }

# define parton container without b-quarks to
```

(continues on next page)

(continued from previous page)

```
# remove 0 processes with top contributions
PARTICLE_CONTAINERS:
  901: {Name: lightflavs, Flavours: [1, -1, 2, -2, 3, -3, 4, -4, 21]}
NLO_CSS_DISALLOW_FLAVOUR: 5

PROCESSES:
- 901 901 -> 90 91 90 91 901{3}:
    Order: {QCD: 0, EW: 4}
    CKKW: 30
    2->4-5:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
      Loop_Generator: OpenLoops
    2->5-7:
      Integration_Error: 0.05

SELECTORS:
- VariableSelector:
    Variable: PT
    Flavs: 90
    Ranges: [[5.0, E_CMS], [5.0, E_CMS]]
    Ordering: [PT_UP]
- [Mass, 11, -11, 10.0, E_CMS]
- [Mass, 13, -13, 10.0, E_CMS]
- [Mass, 15, -15, 10.0, E_CMS]
```

### 9.6.2 Dilepton, missing energy and jets production (gluon initiated)

```
# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# scales
CORE_SCALE: VAR{Abs2(p[2]+p[3]+p[4]+p[5])/4.0}

# me generator settings
ME_GENERATORS: [Amegic, OpenLoops]
AMEGIC: { ALLOW_MAPPING: 0 }
# the following phase space libraries have to be generated with the
# corresponding qq->llvv setup (Sherpa.tree.yaml) first;
# they will appear in Process/Amegic/lib/libProc_fsrchannels*.so
SHERPA_LDADD: [Proc_fsrchannels4, Proc_fsrchannels5]

PROCESSES:
- 93 93 -> 90 90 91 91 93{1}:
    CKKW: $(QCUT)
    Enable_MHV: 10
    Loop_Generator: OpenLoops
    2->4:
      Order: {QCD: 2, EW: 4}
      Integrator: fsrchannels4
    2->5:
      Order: {QCD: 3, EW: 4}
```

(continues on next page)

(continued from previous page)

```
      Integrator: fsrchannels5
      Integration_Error: 0.02

SELECTORS:
- [Mass, 11, -11, 10.0, E_CMS]
- [Mass, 13, -13, 10.0, E_CMS]
- [Mass, 15, -15, 10.0, E_CMS]
```

### 9.6.3 Four lepton and jets production

```
# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# me generator settings
ME_GENERATORS: [Comix, Amegic, OpenLoops]
METS: { CLUSTER_MODE: 16 }

PROCESSES:
- 93 93 -> 90 90 90 90 93{3}:
    Order: {QCD: 0, EW: 4}
    CKKW: 30
    2->4-5:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
      Loop_Generator: OpenLoops
    2->5-7:
      Integration_Error: 0.05

SELECTORS:
- VariableSelector:
    Variable: PT
    Flavs: 90
    Ranges: [[5.0, E_CMS], [5.0, E_CMS]]
    Ordering: [PT_UP]
- [Mass, 11, -11, 10.0, E_CMS]
- [Mass, 13, -13, 10.0, E_CMS]
- [Mass, 15, -15, 10.0, E_CMS]
```

### 9.6.4 Four lepton and jets production (gluon initiated)

```
# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# scales
CORE_SCALE: VAR{Abs2(p[2]+p[3]+p[4]+p[5])/4.0}

# me generator settings
ME_GENERATORS: [Amegic, OpenLoops]
```

(continues on next page)

(continued from previous page)

```
AMEGIC: { ALLOW_MAPPING: 0 }
# the following phase space libraries have to be generated with the
# corresponding qq->llll setup (Sherpa.tree.yaml) first;
# they will appear in Process/Amegic/lib/libProc_fsrchannels*.so
SHERPA_LDADD: [Proc_fsrchannels4, Proc_fsrchannels5]

PROCESSES:
- 93 93 -> 90 90 90 90 93{1}:
    CKKW: $(QCUT)
    Enable_MHV: 10
    Loop_Generator: OpenLoops
    2->4:
      Order: {QCD: 2, EW: 4}
      Integrator: fsrchannels4
    2->5:
      Order: {QCD: 3, EW: 4}
      Integrator: fsrchannels5
      Integration_Error: 0.02

SELECTORS:
- [Mass, 11, -11, 10.0, E_CMS]
- [Mass, 13, -13, 10.0, E_CMS]
- [Mass, 15, -15, 10.0, E_CMS]
```

### 9.6.5 WZ production with jets production

```
# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# scales
CORE_SCALE: VAR{Abs2(p[2]+p[3])/4.0}

# me generator settings
ME_GENERATORS: [Comix, Amegic, OpenLoops]

HARD_DECAYS:
  Enabled: true
  Channels:
    24 -> 2 -1: {Status: 2}
    24 -> 4 -3: {Status: 2}
    -24 -> -2 1: {Status: 2}
    -24 -> -4 3: {Status: 2}
    23 -> 12 -12: {Status: 2}
    23 -> 14 -14: {Status: 2}
    23 -> 16 -16: {Status: 2}

PARTICLE_DATA:
  23: {Width: 0}
  24: {Width: 0}

PROCESSES:
- 93 93 -> 24 23 93{3}:
    Order: {QCD: 0, EW: 2}
    CKKW: 30
```

(continues on next page)

(continued from previous page)

```
    2->2-3:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
      Loop_Generator: OpenLoops
    2->3-7:
      Integration_Error: 0.05
- 93 93 -> -24 23 93{3}:
    Order: {QCD: 0, EW: 2}
    CKKW: 30
    2->2-3:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
      Loop_Generator: OpenLoops
    2->3-7:
      Integration_Error: 0.05
```

### 9.6.6 Same sign dilepton, missing energy and jets production

```
# collider setup
BEAMS: 2212
BEAM_ENERGIES: 6500

# choose EW Gmu input scheme
EW_SCHEME: 3

# tags for process setup
TAGS:
  NJET: 1
  QCUT: 30

# scales
CORE_SCALE: VAR{Abs2(p[2]+p[3]+p[4]+p[5])}
EXCLUSIVE_CLUSTER_MODE: 1

# solves problem with dipole QED modeling
ME_QED: { CLUSTERING_THRESHOLD: 10 }

# improve integration performance
PSI: { ITMIN: 25000 }
INTEGRATION_ERROR: 0.05

PROCESSES:
- 93 93 -> 11 11 -12 -12 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)
- 93 93 -> 13 13 -14 -14 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)
- 93 93 -> 15 15 -16 -16 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)
```

(continues on next page)

(continued from previous page)

```
- 93 93 -> 11 13 -12 -14 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)
- 93 93 -> 11 15 -12 -16 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)
- 93 93 -> 13 15 -14 -16 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)
- 93 93 -> -11 -11 12 12 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)
- 93 93 -> -13 -13 14 14 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)
- 93 93 -> -15 -15 16 16 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)
- 93 93 -> -11 -13 12 14 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)
- 93 93 -> -11 -15 12 16 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)
- 93 93 -> -13 -15 14 16 93 93 93{$(NJET)}:
    Order: {QCD: Any, EW: 6}
    CKKW: $(QCUT)

SELECTORS:
- [PT, 90, 5.0, E_CMS]
- NJetFinder:
    N: 2
    PTMin: 15.0
    ETMin: 0.0
    R: 0.4
    Exp: -1
```

## 9.7 Event generation in the MSSM using UFO

This is an example for event generation in the MSSM using Sherpa's UFO support. In the corresponding Example directory `<prefix>/share/SHERPA-MC/Examples/UFO_MSSM/`, you will find a directory `MSSM` that contains the UFO output for the MSSM (https://feynrules.irmp.ucl.ac.be/wiki/MSSM). To run the example, generate the model as described in *UFO Model Interface* by executing

```
$ cd <prefix>/share/SHERPA-MC/Examples/UFO_MSSM/
$ <prefix>/bin/Sherpa-generate-model MSSM
```

An example run card will be written to the working directory. Use this run card as a template to generate events.

## 9.8 Deep-inelastic scattering



### 9.8.1 DIS at HERA

This is an example of a setup for hadronic final states in deep-inelastic lepton-nucleon scattering at a centre-of-mass energy of 300 GeV. Corresponding measurements were carried out by the H1 and ZEUS collaborations at the HERA collider at DESY Hamburg.

```
# collider setup
BEAMS: [-11, 2212]
BEAM_ENERGIES: [27.5, 820]
PDF_SET: [None, Default]

# technical parameters
TAGS:
  QCUT: 5
  SDIS: 1.0
  LGEN: BlackHat
ME_GENERATORS:
  - Comix
  - Amegic
  - $(LGEN)
RESPECT_MASSIVE_FLAG: true
CSS_KIN_SCHEME: 1

# hadronization tune
PARJ:
  - [21, 0.432]
  - [41, 1.05]
  - [42, 1.0]
  - [47, 0.65]
MSTJ:
  - [11, 5]
FRAGMENTATION: Lund
DECAYMODEL: Lund

PROCESSES:
- -11 93 -> -11 93 93{4}:
    CKKW: $(QCUT)/sqrt(1.0+sqr($(QCUT)/$(SDIS))/Abs2(p[2]-p[0]))
    Order: {QCD: Any, EW: 2}
    Max_N_Quarks: 6
    Loop_Generator: $(LGEN)
    2->2-3:
      NLO_Mode: MC@NLO
      NLO_Order: {QCD: 1, EW: 0}
      ME_Generator: Amegic
      RS_ME_Generator: Comix
    2->3:
      PSI_ItMin: 25000
      Integration_Error: 0.03
```

(continues on next page)

(continued from previous page)

```
SELECTORS:
- [Q2, -11, -11, 4, 1e12]
```

Things to notice:

- the beams are asymmetric with the positrons at an energy of 27.5 GeV, while the protons carry 820 GeV of energy.
- the multi-jet merging cut is set dynamically for each event, depending on the photon virtuality, see [CGH10].
- there is a selector cut on the photon virtuality. This cut implements the experimental requirements for identifying the deep-inelastic scattering process.

## 9.9 Fixed-order next-to-leading order calculations



### 9.9.1 Production of NTuples

Root NTuples are a convenient way to store the result of cumbersome fixed-order calculations in order to perform multiple analyses. This example shows how to generate such NTuples and reweighted them in order to change factorisation and renormalisation scales. Note that in order to use this setup, Sherpa must be configured with option `--enable-root=/path/to/root`, see *Event output formats*. If Sherpa has not been configured with Rivet analysis support, please disable the analysis using `-a0` on the command line, see *Command Line Options*.

When using NTuples, one needs to bear in mind that every calculation involving jets in the final state is exclusive in the sense that a lower cut-off on the jet transverse momenta must be imposed. It is therefore necessary to check whether the event sample stored in the NTuple is sufficiently inclusive before using it. Similar remarks apply when photons are present in the NLO calculation or when cuts on leptons have been applied at generation level to increase efficiency. Every NTuple should therefore be accompanied by an appropriate documentation.

NTuple compression can be customized using the parameter `ROOTNTUPLE_COMPRESSION`, which is used to call `TFile::SetCompressionSettings`. For a detailed documentation of available options, see http://root.cern.ch

This example will generate NTuples for the process pp->lvj, where $l$ is an electron or positron, and $v$ is an electron (anti-)neutrino. We identify parton-level jets using the anti-k_T algorithm with $R=0.4$ [CSS08]. We require the transverse momentum of these jets to be larger than 20 GeV. No other cuts are applied at generation level.

```
EVENTS: 100k
EVENT_GENERATION_MODE: Weighted
TAGS:
  LGEN: BlackHat
ME_GENERATORS: [Amegic, $(LGEN)]
# Analysis (please configure with --enable-rivet & --enable-hepmc2)
ANALYSIS: Rivet
ANALYSIS_OUTPUT: Analysis/HTp/BVI/
# NTuple output (please configure with '--enable-root')
```

(continues on next page)

(continued from previous page)

```
EVENT_OUTPUT: EDRoot[NTuple_B-like]
BEAMS: 2212
BEAM_ENERGIES: 3500
SCALES: VAR{sqr(sqrt(H_T2)-PPerp(p[2])-PPerp(p[3])+MPerp(p[2]+p[3]))/4}
EW_SCHEME: 0
WIDTH_SCHEME: Fixed  # sin\theta_w -> 0.23
DIPOLES: {ALPHA: 0.03}
PARTICLE_DATA:
  13: {Massive: true}
  15: {Massive: true}
PROCESSES:
# The Born piece
- 93 93 -> 90 91 93:
    NLO_Mode: Fixed_Order
    NLO_Part: B
    Order: {QCD: Any, EW: 2}
# The virtual piece
- 93 93 -> 90 91 93:
    NLO_Mode: Fixed_Order
    NLO_Part: V
    Loop_Generator: $(LGEN)
    Order: {QCD: Any, EW: 2}
# The integrated subtraction piece
- 93 93 -> 90 91 93:
    NLO_Mode: Fixed_Order
    NLO_Part: I
    Order: {QCD: Any, EW: 2}
SELECTORS:
- FastjetFinder:
    Algorithm: antikt
    N: 1
    PTMin: 20
    ETMin: 0
    DR: 0.4
RIVET:
  -a: ATLAS_2012_I1083318
  USE_HEPMC_SHORT: 1
  IGNOREBEAMS: 1
```

Things to notice:

- NTuple production is enabled by `EVENT_OUTPUT: Root[NTuple_B-like]`, see *Event output formats*.
- The scale used is defined as in [BBD+09].
- `EW_SCHEME: 0` and `WIDTH_SCHEME: Fixed` are used to set the value of the weak mixing angle to 0.23, consistent with EW precision measurements.
- `DIPOLES:ALPHA: 0.03` is used to limit the active phase space of dipole subtractions.
- `13:Massive: true` and `15:Massive: 1` are used to limit the number of active lepton flavours to electron and positron.
- The option `USE_HEPMC_SHORT: 1` is used in the Rivet analysis section as the events produced by Sherpa are not at particle level.

#### NTuple production

Start Sherpa using the command line

```
$ Sherpa Sherpa.B-like.yaml
```

Sherpa will first create source code for its matrix-element calculations. This process will stop with a message instructing you to compile. Do so by running

```
$ ./makelibs -j4
```

Launch Sherpa again, using

```
$ Sherpa Sherpa.B-like.yaml
```

Sherpa will then compute the Born, virtual and integrated subtraction contribution to the NLO cross section and generate events. These events are analysed using the Rivet library and stored in a Root NTuple file called `NTuple_B-like.root`. We will use this NTuple later to compute an NLO uncertainty band.

The real-emission contribution, including subtraction terms, to the NLO cross section is computed using

```
$ Sherpa Sherpa.R-like.yaml
```

Events are generated, analysed by Rivet and stored in the Root NTuple file `NTuple_R-like.root`.

The two analyses of events with Born-like and real-emission-like kinematics need to be merged, which can be achieved using scripts like `yodamerge`. The result can then be plotted and displayed.

#### Usage of NTuples in Sherpa

Next we will compute the NLO uncertainty band using Sherpa. To this end, we make use of the Root NTuples generated in the previous steps. Note that the setup files for reweighting are almost identical to those for generating the NTuples. We have simply replaced `EVENT_OUTPUT` by `EVENT_INPUT`.

We re-evaluate the events with the scale variation as defined in the `Reweight` configuration files:

```
$ Sherpa Sherpa.Reweight.B-like.yaml
$ Sherpa Sherpa.Reweight.R-like.yaml
```

The contributions can again be combined using `yodamerge`.

### 9.9.2 MINLO

The following configuration file shows how to implement the MINLO procedure from [HNZ]. A few things to note are detailed below. MINLO can also be applied when reading NTuples, see *Production of NTuples*. In this case, the scale and K factor must be defined, see *SCALES* and *KFACTOR*.

```
BEAMS: 2212
BEAM_ENERGIES: 6500

EVENT_GENERATION_MODE: W
CORE_SCALE: VAR{Abs2(p[2]+p[3])+0.25*sqr(sqrt(H_T2)-PPerp(p[2])-
↪PPerp(p[3])+PPerp(p[2]+p[3]))}

PROCESSES:
```

(continues on next page)

(continued from previous page)

```
- 93 93 -> 11 -12 93:
    Scales: MINLO
    KFactor: MINLO
    ME_Generator: Amegic
    Loop_Generator: BlackHat
    Order: {QCD: Any, EW: 2}

SELECTORS:
- [Mass, 11, -12, 2, E_CMS]
- FastjetFinder:
    Algorithm: antikt
    N: 1
    PTMin: 1.0
    ETMin: 1.0
    DR: 0.4
```

Things to notice:

- The R parameter of the flavour-based kT clustering algorithm can be changed using `MINLO:DELTA_R`.
- Setting `MINLO: {SUDAKOV_MODE: 0}` defines whether to include power corrections stemming from the finite parts in the integral over branching probabilities. It defaults to 1.
- The parameter `MINLO:SUDAKOV_PRECISION` defines the precision target for integration of the Sudakov exponent. It defaults to `1e-4`.

## 9.10 Soft QCD: Minimum Bias and Cross Sections



### 9.10.1 Calculation of inclusive cross sections

Note that this example is not yet updated to the new YAML input format. Contact the *Authors* for more information.

```
(run){
  OUTPUT               = 2
  EVENT_TYPE           = MinimumBias
  SOFT_COLLISIONS      = Shrimps
  Shrimps_Mode         = Xsecs

  deltaY     =  1.5;
  Lambda2    =  1.7;
  beta_0^2   =  20.0;
  kappa      =  0.6;
  xi         =  0.2;
  lambda     =  0.3;
  Delta      =  0.4;
}(run)

(beam){
```

(continues on next page)

(continued from previous page)

```
  BEAM_1 =  2212; BEAM_ENERGY_1 = 450.;
  BEAM_2 =  2212; BEAM_ENERGY_2 = 450.;
}(beam)

(me){
  ME_SIGNAL_GENERATOR = None
}(me)
```

Things to notice:

- Inclusive cross sections (total, inelastic, low-mass single-diffractive, low-mass double-diffractive, elastic) and the elastic slope are calculated for varying centre-of-mass energies in pp collisions
- The results are written to the file InclusiveQuantities/xsecs_total.dat and to the screen. The directory will automatically be created in the path from where Sherpa is run.
- The parameters of the model are not very well tuned.

### 9.10.2 Simulation of Minimum Bias events

Note that this example is not yet updated to the new YAML input format. Contact the *Authors* for more information.

```
(run){
  EVENTS               = 50k
  OUTPUT               = 2
  EVENT_TYPE           = MinimumBias
  SOFT_COLLISIONS      = Shrimps
  Shrimps_Mode         = Inelastic

  ANALYSIS             = Rivet

  ANALYSIS_OUTPUT      = test6

  ALPHAS(MZ) 0.118;
  ORDER_ALPHAS 1;
  CSS_FS_PT2MIN        1.00
  MAX_PROPER_LIFETIME = 10.

  deltaY     =  1.5;
  Lambda2    =  1.376;
  beta_0^2   =  18.76;
  kappa      =  0.6;
  xi         =  0.2;
  lambda     =  0.2151;
  Delta      =  0.3052;

  Q_0^2            = 2.25;
  Chi_S            = 1.0;
  Shower_Min_KT2   = 4.0;
  Diff_Factor      = 4.0;
  KT2_Factor       = 4.0;
  RescProb         = 2.0;
  RescProb1        = 0.5;
  Q_RC^2           = 0.9;
  ReconnProb       = -25.;
  Resc_KTMin       = off;
```

(continues on next page)

(continued from previous page)

```
  Misha           = 0
}(run)

(beam){
  BEAM_1 =  2212; BEAM_ENERGY_1 = 3500.;
  BEAM_2 =  2212; BEAM_ENERGY_2 = 3500.;
}(beam)

(analysis){
  BEGIN_RIVET {
  -a ATLAS_2010_S8918562 ATLAS_2010_S8894728 ATLAS_2011_S8994773 ATLAS_2012_I1084540␣
↪TOTEM_2012_DNDETA ATLAS_2011_I919017 CMS_2011_S8978280 CMS_2011_S9120041 CMS_2011_
↪S9215166 CMS_2010_S8656010 CMS_2011_S8884919 CMS_QCD_10_024
  } END_RIVET
}(analysis)

(me){
  ME_SIGNAL_GENERATOR = None
}(me)
```

Things to notice:

- The SHRiMPS model is not properly tuned yet – all parameters are set to very natural values, such as for example 1.0 GeV for infrared parameters.
- Elastic scattering and low-mass diffraction are not included.
- A large number of Minimum Bias-type analyses is enabled.

## 9.11 Setups for event production at B-factories



### 9.11.1 QCD continuum

Example setup for QCD continuum production at the Belle/KEK collider. Please note, it does not include any hadronic resonance.

```
# collider setup
BEAMS:  [11, -11]
BEAM_ENERGIES: [7., 4.]

# model parameters
ALPHAS(MZ): 0.1188
ORDER_ALPHAS: 1
PARTICLE_DATA:
  4: {Massive: 1}
```

(continues on next page)

(continued from previous page)

```
  5: {Massive: 1}
MASSIVE_PS: 3

PROCESSES:
- 11 -11 -> 93 93:
    Order: {QCD: Any, EW: 2}
- 11 -11 -> 4 -4:
    Order: {QCD: Any, EW: 2}
- 11 -11 -> 5 -5:
    Order: {QCD: Any, EW: 2}
```

Things to notice:

- Asymmetric beam energies, photon ISR is switched on per default.
- Full mass effects of c and b quarks computed.
- Strong coupling constant value set to 0.1188 and two loop (NLO) running.

### 9.11.2 Signal process

Example setup for B-hadron pair production on the Y(4S) pole.

```
# collider setup
BEAMS:  [11, -11]
BEAM_ENERGIES: [7., 4.]

# model parameters
ALPHAS(MZ): 0.1188
ORDER_ALPHAS: 1
PARTICLE_DATA:
  4: {Massive: 1}
  5: {Massive: 1}
MASSIVE_PS: 3
ME_GENERATORS: Internal
CORE_SCALE: VAR{sqr(91.2)}

PROCESSES:
#
# electron positron -> Y(4S) -> B+ B-
#
- 11 -11 -> 300553[a]:
    Decay: "300553[a] -> 521 -521"
    Order: {QCD: Any, EW: 2}
#
# electron positron -> Y(4S) -> B0 B0bar
#
- 11 -11 -> 300553[a]:
    Decay: "300553[a] -> 511 -511"
    Order: {QCD: Any, EW: 2}
```

Things to notice:

- Same setup as *QCD continuum*, except for process specification.
- Production of both B0 and B+ pairs, in due proportion.

### 9.11.3 Single hadron decay chains

This setup is not a collider setup, but a simulation of a hadronic decay chain.

```
# collider setup
BEAMS:  [11, -11]
BEAM_ENERGIES: [7., 4.]

# general settings
EVENT_TYPE: HadronDecay

# specify hadron to be decayed
DECAYER: 511

# initialise rest for Sherpa not to complain
# model parameters
ME_GENERATORS: Internal
SCALES: VAR{sqr(91.2)}

PROCESSES:
- 11 -11 -> 13 -13: {}
```

Things to notice:

- `EVENT_TYPE` is set to `HadronDecay`.
- `DECAYER` specifies the hadron flavour initialising the decay chain.
- A place holder process is declared such that the Sherpa framework can be initialised. That process will not be used.

## 9.12 Calculating matrix element values for externally given configurations



### 9.12.1 Computing matrix elements for individual phase space points using the Python Interface

Sherpa's Python interface (see *Python Interface*) can be used to compute matrix elements for individual phase space points. Access to a designated class "MEProcess" is provided by interface to compute matrix elements as illustrated in the example script.

Please note that the process in the script must be compatible with the one specified in the Sherpa configuration file in the working directory. A random phase space point for the process of interest can be generated as shown in the example.

If AMEGIC++ is used as the matrix element generator, executing the script will result in AMEGIC++ writing out libraries and exiting. After compiling the libraries using `./makelibs`, the script must be executed again in order to obtain the matrix element.

```
#!/usr/bin/env python2
@LOADMPIFORPY@
import sys
sys.path.append('@PYLIBDIR@')
import Sherpa

# Add this to the execution arguments to prevent Sherpa from starting the cross␣
↪section integration
sys.argv.append('INIT_ONLY: 2')

Generator=Sherpa.Sherpa(len(sys.argv),sys.argv)
try:
    Generator.InitializeTheRun()
    Process=Sherpa.MEProcess(Generator)

    # Incoming flavors must be added first!
    Process.AddInFlav(11);
    Process.AddInFlav(-11);
    Process.AddOutFlav(1);
    Process.AddOutFlav(-1);
    Process.Initialize();

    # First argument corresponds to particle index:
    # index 0 correspons to particle added first, index 1 is the particle added␣
↪second, and so on...
    Process.SetMomentum(0, 45.6,0.,0.,45.6)
    Process.SetMomentum(1, 45.6,0.,0.,-45.6)
    Process.SetMomentum(2, 45.6,0.,45.6,0.)
    Process.SetMomentum(3, 45.6,0.,-45.6,0.)
    print '\nSquared ME: ', Process.CSMatrixElement()

    # Momentum setting via list of floats
    Process.SetMomenta([[45.6,0.,0.,45.6],
                        [45.6,0.,0.,-45.6],
                        [45.6,0.,45.6,0.],
                        [45.6,0.,-45.6,0.]])
    print '\nSquared ME: ', Process.CSMatrixElement()

    # Random momenta
    E_cms = 500.0
    wgt = Process.TestPoint(E_cms)
    print '\nRandom test point '
    print 'Squared ME: ', Process.CSMatrixElement(), '\n'

except Sherpa.SherpaException as exc:
    print exc
    exit(1)
```

### 9.12.2 Computing matrix elements for individual phase space points using the C++ Interface

Matrix elements values for user defined phase space points can also be quarried using a small C++ executable provided in `Examples/API/ME2`. It can be compiled using the provided `Makefile`. The test program is then run typing (note: the `LD_LIBRARY_PATH` must be set to include `<Sherpa-installation>/lib/SHERPA-MC`)

```
$ ./test <options>
```

where the usual options for Sherpa are passed. An example configuration file, giving both the process and the requested phase space points looks like

```
BEAMS: [11, -11]
BEAM_ENERGIES: 45.6

EVENTS: 0
INIT_ONLY: 2
PDF_LIBRARY: None

PROCESSES:
- 11 -11 -> 2 -2 21 21 21 21: {}

NUMBER_OF_POINTS: 4

MOMENTA:
  - [
      [  11,  45.6,   0.0,   0.0,  45.6 ],
      [ -11,  45.6,   0.0,   0.0, -45.6 ],
      [  21,  10.0,   0.0,   0.0, -10.0, 1, 2 ],
      [  21,  10.0,   0.0,   0.0,  10.0, 2, 3 ],
      [  21,  10.0,  10.0,   0.0,   0.0, 3, 1 ],
      [  21,  10.0, -10.0,   0.0,   0.0, 1, 3 ],
      [   2,  25.6,   0.0,  25.6,   0.0, 3, 0 ],
      [  -2,  25.6,   0.0, -25.6,   0.0, 0, 1 ]
    ]
  - [
      [  11,  45.6,   0.0,   0.0,  45.6 ],
      [ -11,  45.6,   0.0,   0.0, -45.6 ],
      [  21,  12.0,   0.0,   0.0, -12.0, 1, 2 ],
      [  21,  12.0,   0.0,   0.0,  12.0, 2, 3 ],
      [  21,  12.0,  12.0,   0.0,   0.0, 3, 1 ],
      [  21,  12.0, -12.0,   0.0,   0.0, 1, 3 ],
      [   2,  21.6,   0.0,  21.6,   0.0, 3, 0 ],
      [  -2,  21.6,   0.0, -21.6,   0.0, 0, 1 ]
    ]
  - [
      [  11,  45.6,   0.0,   0.0,  45.6 ],
      [ -11,  45.6,   0.0,   0.0, -45.6 ],
      [  21,  14.0,   0.0,   0.0, -14.0, 1, 2 ],
      [  21,  14.0,   0.0,   0.0,  14.0, 2, 3 ],
      [  21,  14.0,  14.0,   0.0,   0.0, 3, 1 ],
      [  21,  14.0, -14.0,   0.0,   0.0, 1, 3 ],
      [   2,  17.6,   0.0,  17.6,   0.0, 3, 0 ],
      [  -2,  17.6,   0.0, -17.6,   0.0, 0, 1 ]
    ]
  - [
      [  11,  45.6,   0.0,   0.0,  45.6 ],
```

(continues on next page)

(continued from previous page)

```
      [ -11,  45.6,   0.0,   0.0, -45.6 ],
      [  21,  16.0,   0.0,   0.0, -16.0, 1, 2 ],
      [  21,  16.0,   0.0,   0.0,  16.0, 2, 3 ],
      [  21,  16.0,  16.0,   0.0,   0.0, 3, 1 ],
      [  21,  16.0, -16.0,   0.0,   0.0, 1, 3 ],
      [   2,  13.6,   0.0,  13.6,   0.0, 3, 0 ],
      [  -2,  13.6,   0.0, -13.6,   0.0, 0, 1 ]
    ]
```

Please note that both the process and the beam specifications need to be present in order for Sherpa to initialise properly. The momenta need to be given in the proper ordering employed in Sherpa, which can be read from the process name printed on screen. For each entry the sequence is the following

```
[<pdg-id>, <E>, <px>, <py>, <pz>, triplet-index, antitriplet-index]
```

with the colour indices ranging from 1..3 for both the triplet and the antitriplet index in the colour-flow basis. The colour information is only needed if Comix is used for the calculation as Comix then also gives the squared matrix element value for this colour configuration. Otherwise, the last two arguments can be omitted. In any case, the colour-summed value is printed to screen.

## 9.13 Using the Python interface



### 9.13.1 Generate events using scripts

This example shows how to generate events with Sherpa using a Python wrapper script. For each event the weight, the number of trials and the particle information is printed to stdout. This script can be used as a basis for constructing interfaces to own analysis routines.

```
#!/usr/bin/python2
@LOADMPIFORPY@
import sys
sys.path.append('@PYLIBDIR@')
import Sherpa

Generator=Sherpa.Sherpa(len(sys.argv),sys.argv)
try:
    Generator.InitializeTheRun()
    Generator.InitializeTheEventHandler()
    for n in range(1,1+Generator.NumberOfEvents()):
        Generator.GenerateOneEvent()
        blobs=Generator.GetBlobList();
        print "Event",n,"{"
        ## print blobs
        print "  Weight ",blobs.GetFirst(1)["Weight"];
        print "  Trials ",blobs.GetFirst(1)["Trials"];
        for i in range(0,blobs.size()):
```

(continues on next page)

(continued from previous page)

```
            print "  Blob",i,"{"
            ## print blobs[i];
            print "    Incoming particles"
            for j in range(0,blobs[i].NInP()):
                part=blobs[i].InPart(j)
                ## print part
                s=part.Stat()
                f=part.Flav()
                p=part.Momentum()
                print "     ",j,": ",s,f,p
            print "    Outgoing particles"
            for j in range(0,blobs[i].NOutP()):
                part=blobs[i].OutPart(j)
                ## print part
                s=part.Stat()
                f=part.Flav()
                p=part.Momentum()
                print "     ",j,": ",s,f,p
            print "  } Blob",i
        print "} Event",n
        if ((n%100)==0): print "  Event ",n
    Generator.SummarizeRun()

except Sherpa.SherpaException as exc:
    exit(1)
```

### 9.13.2 Generate events with MPI using scripts

This example shows how to generate events with Sherpa using a Python wrapper script and MPI. For each event the weight, the number of trials and the particle information is send to the MPI master node and written into a single gzip'ed output file. Note that you need the mpi4py module to run this Example. Sherpa must be configured and installed using `--enable-mpi`, see *MPI parallelization*.

```
#!/usr/bin/python2
@LOADMPIFORPY@
import sys
sys.path.append('@PYLIBDIR@')
import Sherpa
import gzip

class MyParticle:
    def __init__(self,p):
        self.kfc=p.Flav().Kfcode()
        if p.Flav().IsAnti(): self.kfc=-self.kfc
        self.E=p.Momentum()[0]
        self.px=p.Momentum()[1]
        self.py=p.Momentum()[2]
        self.pz=p.Momentum()[3]
    def __str__(self):
        return (str(self.kfc)+" "+str(self.E)+" "
                +str(self.px)+" "+str(self.py)+" "+str(self.pz))

Generator=Sherpa.Sherpa(len(sys.argv),sys.argv)
try:
    Generator.InitializeTheRun()
```

(continues on next page)

(continued from previous page)

```
    Generator.InitializeTheEventHandler()
    comm=MPI.COMM_WORLD
    rank=comm.Get_rank()
    size=comm.Get_size()
    if rank==0:
        outfile=gzip.GzipFile("events.gz",'w')
        for n in range(1,1+Generator.NumberOfEvents()):
            for t in range(1,size):
                weight=comm.recv(source=t,tag=t)
                trials=comm.recv(source=t,tag=2*t)
                parts=comm.recv(source=t,tag=3*t)
                outfile.write("E "+str(weight)+" "+str(trials)+"\n")
                for p in parts:
                    outfile.write(str(p)+"\n")
            if (n%100)==0: print "  Event",n
        outfile.close()
    else:
        for n in range(1,1+Generator.NumberOfEvents()):
            Generator.GenerateOneEvent()
            blobs=Generator.GetBlobList();
            weight=blobs.GetFirst(1)["Weight"]
            trials=blobs.GetFirst(1)["Trials"]
            parts=[]
            for i in range(0,blobs.size()):
                for j in range(0,blobs[i].NOutP()):
                    part=blobs[i].OutPart(j)
                    if part.Stat()==1 and part.HasDecBlob()==0:
                        parts.append(MyParticle(part))
            comm.send(weight,dest=0,tag=rank)
            comm.send(trials,dest=0,tag=2*rank)
            comm.send(parts,dest=0,tag=3*rank)
    Generator.SummarizeRun()

except Sherpa.SherpaException as exc:
    exit(1)
```

CHAPTER

# TEN

# GETTING HELP

If Sherpa exits abnormally, first check the Sherpa output for hints on the reason of program abort, and try to figure out what has gone wrong with the help of the Manual. Note that Sherpa throwing a `normal_exit` exception does not imply any abnormal program termination! When using AMEGIC++ Sherpa will exit with the message:

```
New libraries created. Please compile.
```

In this case, follow the instructions given in *Running Sherpa with AMEGIC++*.

If this does not help, contact the Sherpa team (see the Sherpa Team section of the website sherpa.hepforge.org), providing all information on your setup. Please include

1. A complete tarred and gzipped set of the `Sherpa.yaml` config file leading to the crash. Use the status recovery directory `Status__<date of crash>` produced before the program abort.
2. The command line (including possible parameters) you used to start Sherpa.
3. The installation log file, if available.

CHAPTER

# ELEVEN

# AUTHORS

Sherpa was written by the Sherpa Team, see sherpa-team.gitlab.io.

CHAPTER

TWELVE

# COPYING

Sherpa is free software. You can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation. You should have received a copy of the GNU General Public License along with the source for Sherpa; see the file COPYING. If not, write to the Free Software Foundation, 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA.

Sherpa is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Sherpa was created during the Marie Curie RTN's HEPTOOLS, MCnet and LHCphenonet. The MCnet Guidelines apply, see the file GUIDELINES and https://www.montecarlonet.org/guidelines.

CHAPTER

THIRTEEN

# APPENDIX A: REFERENCES

- genindex

# BIBLIOGRAPHY


[A+a] S. Alekhin and others. Hera and the lhc - a workshop on the implications of hera for lhc physics: proceedings part a. :. hep-ph/0601012.

[A+b] S. Alioli and others. Update of the binoth les houches accord for a standard interface between monte carlo tools and one-loop programs. :. arXiv:1308.3462.

[A+08] J. Alwall and others. Comparative study of various algorithms for the merging of parton showers and matrix elements in hadronic collisions. *Eur. Phys. J.*, C53:473–500, 2008. arXiv:0706.2569.

[AGH+08] J. Archibald, T. Gleisberg, S. Hoche, F. Krauss, M. Schonherr, S. Schumann, F. Siegert, and J. C. Winter. Simulation of photon-photon interactions in hadron collisions with sherpa. *Nucl. Phys.*, 179:218–225, 2008. F+J+Nucl. Phys.

[B+08] M. Bahr and others. Herwig++ physics and manual. *Eur. Phys. J.*, C58:639–707, 2008. arXiv:0803.0883.

[B+] R. D. Ball and others. Parton distributions for the lhc run ii. :. [arXiv:1410.8849].

[BMM94] A. Ballestrero, E. Maina, and S. Moretti. Heavy quarks and leptons at e^+e^- colliders. *Nucl. Phys.*, B415:265–292, 1994. hep-ph/9212246.

[Ba94] E. Barberio and Z. Wc as. Photos - a universal monte carlo for qed radiative corrections: version 2.0. *Comput. Phys. Commun.*, 79:291–308, 1994. F+J+Comput. Phys. Commun.

[Bea03] D. M. Beazley. Automated scientific software scripting with swig. *Future Generation Computer Systems*, 19:599–609, 2003.

[BPK94] F. A. Berends, R. Pittau, and R. Kleiss. All electroweak four-fermion processes in electron-positron collisions. *Nucl. Phys.*, B424:308, 1994. hep-ph/9404313.

[BBD+09] C. F. Berger, Z. Bern, L. J. Dixon, F. Febres-Cordero, D. Forde, T. Gleisberg, H. Ita, D. A. Kosower, and D. Ma^itre. Next-to-leading order qcd predictions for w+3-jet distributions at hadron colliders. *Phys. Rev.*, D80:074036, 2009. arXiv:0907.1984.

[BDS02] Z. Bern, L. J. Dixon, and C. Schmidt. Isolating a light higgs boson from the di-photon background at the lhc. *Phys. Rev.*, D66:074018, 2002. hep-ph/0206194.

[B+10] T. Binoth and others. A proposal for a standard interface between monte carlo tools and one-loop programs. *Comput. Phys. Commun.*, 181:1612–1622, 2010. arXiv:1001.1307.

[BSK] E. Bothmann, M. Schonherr, and F. Krauss. Single top-quark production with sherpa. :. [arXiv:1711.02568].

[B+19] Enrico Bothmann and others. Event Generation with Sherpa 2.2. *SciPost Phys.*, 7(3):034, 2019. arXiv:1905.09127, doi:10.21468/SciPostPhys.7.3.034.

[BGMS74] V. M. Budnev, I. F. Ginzburg, G. V. Meledin, and V. G. Serbo. The two photon particle production mechanism. physical problems. applications. equivalent photon approximation. *Phys. Rept.*, 15:181–281, 1974. F+J+Phys. Rept.

[B+15] M. Buschmann and others. Mass effects in the higgs-gluon coupling: boosted vs off-shell production. *JHEP*, 1502:038, 2015. [hep-ph/1410.5806].

[CSS08] M. Cacciari, G. P. Salam, and G. Soyez. The anti-k(t) jet clustering algorithm. *JHEP*, 0804:063, 2008. arXiv:0802.1189.

[CGH10] T. Carli, T. Gehrmann, and S. Hoche. Hadronic final states in deep-inelastic scattering with sherpa. *Eur. Phys. J.*, C67:73, 2010. arXiv:0912.3715.

[CMP12] F. Cascioli, P. Maierhofer, and S. Pozzorini. Scattering amplitudes with open loops. *Eur.Phys.J.*, C72:1889, 2012. arXiv:1111.5206.

[CDST02] S. Catani, S. Dittmaier, M. H. Seymour, and Z. Trocsanyi. The dipole formalism for next-to-leading order qcd calculations with massive partons. *Nucl. Phys.*, B627:189–265, 2002. hep-ph/0201036.

[CKKW01] S. Catani, F. Krauss, R. Kuhn, and B. R. Webber. Qcd matrix elements + parton showers. *JHEP*, 11:063, 2001. hep-ph/0109231.

[CS97] S. Catani and M. H. Seymour. A general algorithm for calculating jet cross sections in nlo qcd. *Nucl. Phys.*, B485:291–419, 1997. hep-ph/9605323.

[CdAD+11] N. D. Christensen, P. de Aquino, C. Degrande, C. Duhr, B. Fuks, M. Herquet, F. Maltoni, and S. Schumann. A comprehensive approach to new physics simulations. *Eur. Phys. J.*, C71:1541, 2011. arXiv:0906.2474.

[CD09] N. D. Christensen and C. Duhr. Feynrules - feynman rules made easy. *Comput. Phys. Commun.*, 180:1614–1641, 2009. arXiv:0806.4194.

[DDF+12] C. Degrande, C. Duhr, B. Fuks, D. Grellscheid, O. Mattelaer, and T. Reiter. Ufo - the universal feynrules output. *Comput. Phys. Commun.*, 138:1201, 2012. arXiv:1108.2040.

[DL] L. J. Dixon and Y. Li. Bounding the higgs boson width through interferometry. :. arXiv:1305.3854.

[DvHK00] P. D. Draggiotis, A. van Hameren, and R. Kleiss. Sarge: an algorithm for generating qcd-antennas. *Phys. Lett.*, B483:124–130, 2000. hep-ph/0004047.

[DHM06] C. Duhr, S. Hoche, and F. Maltoni. Color-dressed recursive relations for multi-parton amplitudes. *JHEP*, 08:062, 2006. hep-ph/0607057.

[D+] S. Dulat and others. New parton distribution functions from a global analysis of quantum chromodynamics. :. [arXiv:1506.07443].

[FW83] R. D. Field and S. Wolfram. A qcd model for e^+e^- annihilation. *Nucl. Phys.*, B213:65, 1983. F+J+Nucl. Phys.

[Fri98] S. Frixione. Isolated photons in perturbative qcd. *Phys. Lett.*, B429:369–374, 1998. hep-ph/9801442.

[GGH+] J. Gao, M. Guzzi, J. Huston, H. L. Lai, Z. Li, and others. The ct10 nnlo global analysis of qcd. :. arXiv:1302.6246.

[GHK+13] T. Gehrmann, S. Hoche, F. Krauss, M. Schonherr, and F. Siegert. Nlo qcd matrix elements + parton showers in e^+e^- -> hadrons. *JHEP*, 01:144, 2013. arXiv:1207.5031.

[GH08] T. Gleisberg and S. Hoche. Comix a new matrix element generator. *JHEP*, 12:039, 2008. arXiv:0808.3674.

[GHK+09] T. Gleisberg, S. Hoche, F. Krauss, M. Schonherr, S. Schumann, F Siegert, and J. Winter. Event generation with sherpa 1.1. *JHEP*, 02:007, 2009. arXiv:0811.4622.

[GK08] T. Gleisberg and F. Krauss. Automating dipole subtraction for qcd nlo calculations. *Eur. Phys. J.*, C53:501–523, 2008. arXiv:0709.2881.

[GKP+04] T. Gleisberg, F. Krauss, C. G. Papadopoulos, A. Schalicke, and S. Schumann. Cross sections for multi-particle final states at a linear collider. *Eur. Phys. J.*, C34:173–180, 2004. hep-ph/0311273.

[GKS+05] T. Gleisberg, F. Krauss, A. Schalicke, S. Schumann, and J. C. Winter. Studying w^+ w^- production at the fermilab tevatron with sherpa. *Phys. Rev.*, D72:034028, 2005. hep-ph/0504032.

[GRV92a] M. Gluck, E. Reya, and A. Vogt. Parton structure of the photon beyond the leading order. *Phys. Rev.*, D45:3986–3994, 1992. F+J+Phys. Rev.

[GRV92b] M. Gluck, E. Reya, and A. Vogt. Photonic parton distributions. *Phys. Rev.*, D46:1973–1979, 1992. F+J+Phys. Rev.

[Got83] T. D. Gottschalk. A realistic model for e^+e^- annihilation including parton bremsstrahlung effects. *Nucl. Phys.*, B214:201, 1983. F+J+Nucl. Phys.

[Got84] T. D. Gottschalk. An improved description of hadronization in the qcd cluster model for e^+e^- annihilation. *Nucl. Phys.*, B239:349, 1984. F+J+Nucl. Phys.

[GM87] T. D. Gottschalk and D. A. Morris. A new model for hadronization and e^+e^- annihilation. *Nucl. Phys.*, B288:729, 1987. F+J+Nucl. Phys.

[H+06] K. Hagiwara and others. Supersymmetry simulations with off-shell effects for the cern lhc and an ilc. *Phys. Rev.*, D73:055005, 2006. hep-ph/0512260.

[HN10] K. Hamilton and P. Nason. Improving nlo-parton shower matched simulations with higher order matrix elements. *JHEP*, 06:039, 2010. arXiv:1004.1764.

[HNZ] K. Hamilton, P. Nason, and G. Zanderighi. Minlo: multi-scale improved nlo. :. arXiv:1206.3572.

[HRT09] K. Hamilton, P. Richardson, and J. Tully. A modified ckkw matrix element merging approach to angular-ordered parton showers. *JHEP*, 11:038, 2009. arXiv:0905.3072.

[HKSS11] S. Hoche, F. Krauss, M. Schonherr, and F. Siegert. Nlo matrix elements and truncated showers. *JHEP*, 08:123, 2011. arXiv:1009.1127.

[HKSS12] S. Hoche, F. Krauss, M. Schonherr, and F. Siegert. A critical appraisal of nlo+ps matching methods. *JHEP*, 09:049, 2012. arXiv:1111.1220.

[HKSS13] S. Hoche, F. Krauss, M. Schonherr, and F. Siegert. Qcd matrix elements + parton showers: the nlo case. *JHEP*, 04:027, 2013. arXiv:1207.5030.

[HKSS09] S. Hoche, F. Krauss, S. Schumann, and F. Siegert. Qcd matrix elements and truncated showers. *JHEP*, 05:053, 2009. arXiv:0903.1219.

[H+] S. Hoche and others. Matching parton showers and matrix elements. :. hep-ph/0602031.

[HS12] S. Hoche and M. Schonherr. Uncertainties in nlo + parton shower matched simulations of inclusive jet and dijet production. *Phys.Rev.*, D86:094042, 2012. arXiv:1208.2815.

[HSS10] S. Hoche, S. Schumann, and F. Siegert. Hard photon production and matrix-element parton-shower merging. *Phys. Rev.*, D81:034026, 2010. arXiv:0912.3501.

[HHL+13] S. Hoeche, J. Huang, G. Luisoni, M. Schoenherr, and J. Winter. Zero and one jet combined nlo analysis of the top quark forward-backward asymmetry. *Phys.Rev.*, D88:014040, 2013. arXiv:1306.2703.

[HKP+] S. Hoeche, F. Krauss, S. Pozzorini, M. Schoenherr, J. M. Thompson, and K. C. Zapp. Triple vector boson production through higgs-strahlung with nlo multijet merging. :. arXiv:1403.7516.

[HKS] S. Hoeche, F. Krauss, and M. Schoenherr. Uncertainties in meps@@nlo calculations of h+jets. :. arXiv:1401.7971.

[HKSS15] S. Hoeche, S. Kuttimalai, S. Schumann, and F. Siegert. Beyond standard model calculations with sherpa. *Eur. Phys. J.*, C75:135, 2015. arXiv:1412.6478.

[HP] S. Hoeche and S. Prestel. The midpoint between dipole and parton showers. :. arXiv:1506.05057.

[JWDK93] S. Jadach, Z. Was, R. Decker, and J. H. Kuhn. The tau decay library tauola: version 2.4. *Comput. Phys. Commun.*, 76:361–380, 1993. F+J+Comput. Phys. Commun.

[KP00] A. Kanaki and C. G. Papadopoulos. Helac: a package to compute electroweak helicity amplitudes. *Comput. Phys. Commun.*, 132:306–315, 2000. hep-ph/0002082.

[KP94] R. Kleiss and R. Pittau. Weight optimization in multichannel monte carlo. *Comput. Phys. Commun.*, 83:141–146, 1994. hep-ph/9405257.

[KS85] R. Kleiss and W. J. Stirling. Spinor techniques for calculating pbarpto w^pm/z^0+jets. *Nucl. Phys.*, B262:235–262, 1985. F+J+Nucl. Phys.

[KSE86] R. Kleiss, W. J. Stirling, and S. D. Ellis. A new monte carlo treatment of multiparticle phase space at high energies. *Comput. Phys. Commun.*, 40:359, 1986. F+J+Comput. Phys. Commun.

[Kra02] F. Krauss. Matrix elements and parton showers in hadronic interactions. *JHEP*, 0208:015, 2002. hep-ph/0205283.

[KKS02] F. Krauss, R. Kuhn, and G. Soff. Amegic++ 1.0: a matrix element generator in c++. *JHEP*, 02:044, 2002. hep-ph/0109036.

[KLS] F. Krauss, T. Laubrich, and F. Siegert. Simulation of hadron decays in sherpa. :.

[KSSS04] F. Krauss, A. Schalicke, S. Schumann, and G. Soff. Simulating w/z + jets production at the tevatron. *Phys. Rev.*, D70:114009, 2004. hep-ph/0409106.

[KSSS05] F. Krauss, A. Schalicke, S. Schumann, and G. Soff. Simulating w/z + jets production at the cern lhc. *Phys. Rev.*, D72:054017, 2005. hep-ph/0503280.

[LGH+10] H. L. Lai, M. Guzzi, J. Huston, Z. Li, P. M. Nadolsky, and others. New parton distributions for collider physics. *Phys.Rev.*, D82:074024, 2010. arXiv:1007.2241.

[Lan01] D. J. Lange. The evtgen particle decay simulation package. *Nucl. Instrum. Meth.*, A462:152–155, 2001. F+J+Nucl. Instrum. Meth.

[LL05] N. Lavesson and L. Lonnblad. W+jets matrix elements and the dipole cascade. *JHEP*, 07:054, 2005. hep-ph/0503293.

[LL08] N. Lavesson and L. Lonnblad. Extending ckkw-merging to one-loop matrix elements. *JHEP*, 12:070, 2008. arXiv:0811.2912.

[Lep] G. P. Lepage. Vegas - an adaptive multi-dimensional integration program. :.

[Lon92] L. Lonnblad. Ariadne version 4: a program for simulation of qcd cascades implementing the colour dipole model. *Comput. Phys. Commun.*, 71:15–31, 1992. F+J+Comput. Phys. Commun.

[Lon02] L. Lonnblad. Correcting the colour-dipole cascade model with fixed order matrix elements. *JHEP*, 05:046, 2002. hep-ph/0112284.

[LPa] L. Lonnblad and S. Prestel. Merging multi-leg nlo matrix elements with parton showers. :. arXiv:1211.7278.

[LPb] L. Lonnblad and S. Prestel. Unitarising matrix element + parton shower merging. :. arXiv:1211.4827.

[LP12] L. Lonnblad and S. Prestel. Matching tree-level matrix elements with interleaved showers. *JHEP*, 03:019, 2012. arXiv:1109.4829.

[MS03] F. Maltoni and T. Stelzer. Madevent: automatic event generation with madgraph. *JHEP*, 02:027, 2003. hep-ph/0208156.

[MMP+03] M. L. Mangano, M. Moretti, F. Piccinini, R. Pittau, and A. D. Polosa. Alpgen a generator for hard multiparton processes in hadronic collisions. *JHEP*, 07:001, 2003. hep-ph/0206293.

[MMPT07] M. L. Mangano, M. Moretti, F. Piccinini, and M. Treccani. Matching matrix elements and shower evolution for top-pair production in hadronic collisions. *JHEP*, 01:013, 2007. hep-ph/0611129.

[MMP02] M. L. Mangano, M. Moretti, and R. Pittau. Multijet matrix elements and shower evolution in hadronic collisions: w bbarb+n-jets as a case study. *Nucl. Phys.*, B632:343–362, 2002. hep-ph/0108069.

[MW88] G. Marchesini and B. R. Webber. Monte carlo simulation of general hard processes with coherent qcd radiation. *Nucl. Phys.*, B310:461, 1988. F+J+Nucl. Phys.

[MZ91] George Marsaglia and Arif Zaman. A new class of random number generators. *Ann. Appl. Probab.*, 1(3):462–480, 08 1991. URL: https://doi.org/10.1214/aoap/1177005878, doi:10.1214/aoap/1177005878.

[MRST00] A. D. Martin, R. G. Roberts, W. J. Stirling, and R. S. Thorne. Parton distributions and the lhc: w and z production. *Eur. Phys. J.*, C14:133–145, 2000. hep-ph/9907231.

[MRST02] A. D. Martin, R. G. Roberts, W. J. Stirling, and R. S. Thorne. Mrst2001: partons and alpha_s from precise deep inelastic scattering and tevatron jet data. *Eur. Phys. J.*, C23:73–87, 2002. hep-ph/0110215.

[MRST05] A. D. Martin, R. G. Roberts, W. J. Stirling, and R. S. Thorne. Parton distributions incorporating qed contributions. *Eur. Phys. J.*, C39:155–161, 2005. hep-ph/0411040.

[MSTW09] A. D. Martin, W. J. Stirling, R. S. Thorne, and G. Watt. Parton distributions for the lhc. *Eur. Phys. J.*, C63:189–295, 2009. arXiv:0901.0002.

[N+08] P. M. Nadolsky and others. Implications of cteq global analysis for collider observables. *Phys. Rev.*, D78:013004, 2008. arXiv:0802.0007.

[Nag03] Z. Nagy. Next-to-leading order calculation of three-jet observables in hadron-hadron collisions. *Phys. Rev.*, D68:094002, 2003. hep-ph/0307268.

[NS] Z. Nagy and D. E. Soper. A new parton shower algorithm: shower evolution matching at leading and next-to-leading order level. :. hep-ph/0601021.

[NS05] Z. Nagy and D. E. Soper. Matching parton showers to nlo computations. *JHEP*, 10:024, 2005. hep-ph/0503053.

[PW07] C. G. Papadopoulos and M. Worek. Multi-parton cross sections at hadron colliders. *Eur. Phys. J.*, C50:843–856, 2007. hep-ph/0512150.

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition - The Art of Scientific Computing*. Cambridge University Press, Cambridge, 3. aufl. edition, 2007. ISBN 978-0-521-88068-8.

[RMK09] M. G. Ryskin, A. D. Martin, and V. A. Khoze. Soft processes at the lhc i: multi-component model. *Eur. Phys. J.*, C60:249–264, 2009. arXiv:0812.2407.

[SK08a] M. Schonherr and F. Krauss. Soft photon radiation in particle decays in sherpa. *JHEP*, 12:018, 2008. arXiv:0810.5071.

[SK08b] S. Schumann and F. Krauss. A parton shower algorithm based on catani-seymour dipole factorisation. *JHEP*, 03:038, 2008. arXiv:0709.1027.

[SMS06] T. Sjostrand, S. Mrenna, and P. Skands. Pythia 6.4 physics and manual. *JHEP*, 05:026, 2006. hep-ph/0603175.

[SMS08] T. Sjostrand, S. Mrenna, and P. Skands. A brief introduction to pythia 8.1. *Comput. Phys. Commun.*, 178:852–867, 2008. arXiv:0710.3820.

[SvZ87] T. Sjostrand and M. van Zijl. A multiple-interaction model for the event structure in hadron collisions. *Phys. Rev.*, D36:2019, 1987. F+J+Phys. Rev.

[SL94] T. Stelzer and W. F. Long. Automatic generation of tree level helicity amplitudes. *Comput. Phys. Commun.*, 81:357–371, 1994. hep-ph/9401258.

[SJW10] I. W. Stewart, F. J. Tackmann W. J., and Waalewijn. N-jettiness: an inclusive event shape to veto jets. *Phys. Rev. Lett.*, 105:092002, 2010. arXiv:1004.2489.

[vHP02] A. van Hameren and C. G. Papadopoulos. A hierarchical phase space generator for qcd antenna structures. *Eur. Phys. J.*, C25:563–574, 2002. hep-ph/0204055.

[Web84] B. R. Webber. A qcd model for jet fragmentation including soft gluon interference. *Nucl. Phys.*, B238:492, 1984. F+J+Nucl. Phys.

[WBG] M. R. Whalley, D. Bourilkov, and R. C. Group. The les houches accord pdfs (lhapdf) and lhaglue. :. hep-ph/0508110.

[WKS04] J. C. Winter, F. Krauss, and G. Soff. A modified cluster-hadronisation model. *Eur. Phys. J.*, C36:381–395, 2004. hep-ph/0311085.

[Wol83] L. Wolfenstein. Parametrization of the kobayashi-maskawa matrix. *Phys. Rev. Lett.*, 51:1945, 1983. F+J+Phys. Rev. Lett.

[YFS61] D. R. Yennie, S. C. Frautschi, and H. Suura. The infrared divergence phenomena and high-energy processes. *Ann. Phys.*, 13:379–452, 1961. F+J+Ann. Phys.

[Zar03] A. F. Zarnecki. Compaz: parametrization of the luminosity spectra for the photon collider. *Acta Phys. Polon.*, B34:2741–2758, 2003. hep-ex/0207021.

# INDEX

## Symbols

## A



## B



## C

## D



## E



## F



## G



## H



## I



## K



## L



## M

## N



## O



## P



## R



## S

## T



## U



## V



## W



## X



## Y