LC-TOOL-2001-039
(revised)

WHIZARD 1.97

A generic Monte-Carlo integration and event generation package for multi-particle processes

MANUAL [1]

Wolfgang Kilian,[2] Thorsten Ohl,[3] Jürgen Reuter,[4] Christian Speckner,[5]

Universität Siegen, D–57068 Siegen, Germany
Universität Würzburg, D–97074 Würzburg, Germany
Deutsches Elektronen-Synchrotron DESY, D–22603 Hamburg, Germany
Universität Freiburg, D–79104 Freiburg, Germany

[1]Work supported by the Helmholtz-Gemeinschaft, the BMBF etc.
[2]e-mail: wolfgang.kilian@desy.de
[3]e-mail: ohl@physik.uni-wuerzburg.de
[4]e-mail: juergen.reuter@desy.de
[5]e-mail: christian.speckner@physik.uni-freiburg.de

ABSTRACT

WHIZARD is a program system designed for the efficient calculation of multi-particle scattering cross sections and simulated event samples. The events can be written to file in various formats (including STDHEP and ASCII) or analyzed directly on the parton level using a built-in LaTeX-compatible graphics package.

Tree-level matrix elements are generated automatically for arbitrary partonic processes by calling (in principle) external programs (O'Mega per default, while MadGraph and CompHEP are available on demand), that are however shipped with the main distribution. In particular, the MSSM is supported with an interface to the SUSY Les Houches Accord input format. Matrix elements obtained by alternative methods (e.g., including loop corrections) may be interfaced as well. Using an adaptive multi-channel method for phase space integration, the program is able to calculate numerically stable signal and background cross sections and generate unweighted event samples with reasonable efficiency for processes with up to six and more final-state particles. Polarization is treated exactly for both the initial and final states. Quark or lepton flavors can be summed over automatically where needed.

For Linear Collider physics, beamstrahlung (CIRCE), Compton and ISR spectra are included for electrons and photons. Alternatively, beam-crossing events can be read directly from file. For hadron collider physics, the standard LHAPDF (or alternatively for backwards compatibility reasons) the PDF library (PDFLIB) can be linked. For fragmenting and hadronizing the final state, a PYTHIA and Herwig interface is provided which follows the Les Houches Accord.

The WHIZARD distribution is available at

http://projects.hepforge.org/whizard

or alternatively from

http://whizard.event-generator.org

Contents

1 General remarks and History

The increase of complexity in the scattering processes that are of interest at the next generation of colliders calls for new tools for automatic computation and event generation. Many different problems have to be tackled simultaneously: The program has to be versatile and necessarily involves a large degree of automatization since the list of multi-particle processes to be considered is much longer than can be included in a hard-coded process library like `PYTHIA` [1]. The precision required for the predictions makes the traditional distinction between signal and background processes obsolete, since interference effects often cannot be neglected. Thus, massive multi-particle phase space has to be handled in the presence of many resonances and nearby singularities. Since detector effects need to be studied, the program has to have a convenient user interface, and it must be able to generate unweighted events with reasonable efficiency.

During the workshops of the ECFA/DESY study for a future Linear Collider it became obvious that no single existing package was able to meet all these needs. Therefore, the initial idea of `WHIZARD` was to combine known packages for generating matrix elements with a program which is able to treat generic phase space, integrate and generate events. The included matrix element generator is `O'Mega` [4] (for historic reasons and for internal compatibility and cross checks) slightly modified late 1990s' versions of `CompHEP` [2] and `MadGraph` [3] are also available). `O'Mega` covers the whole set of processes that currently can be handled automatically (at tree level). The latter problem could be solved with the help of the `VAMP` [5] integration program, which extends the `VEGAS` algorithm to multi-channel parameterizations and thus makes it possible to handle the complex singularity patterns of multi-particle phase space in a uniform way.

The task for `WHIZARD` was to provide the actual phase space parameterizations, Jacobians and transformations, provide a consistent environment and to make the programs communicate with each other by common interfaces. The user had to be given a simple setup with common configuration and parameter definition files, commands to run all programs consistently without the need for manual intervention (a simple `make` should suffice), and an analysis system which allows for rapid inspection of the results as well as for interfacing hadronization and detector simulation programs. The program had to keep full track of beam polarization and include beamstrahlung and initial-state radiation. For hadronization, an interface to external programs should be included. Finally, it should allow for flavor summation in the final state: usually, many different processes contribute to a single final state that cannot be distinguished experimentally, and thus should be covered in a single run.

These goals have to a large extent been achieved by the current release. However, there is still room for improvement. Most prominent, for a longer time period, parts of the support for QCD amplitudes was given by the `O'Mega` matrix element generator only in the leading N_c approximation. This problem has been successfully solved with the 1.41 release in July 2005. However, the treatment of QCD effects still is (and probably will ever be) incomplete, which always is a problem of the matrix element generators, the shower generators and the hadronization models which e.g. do not treat interfering color structures correctly in all cases. Furthermore, these programs obviously limit `WHIZARD` to the physics models they can support.

The original acronym `WHIZARD` stands for

which is the class of processes (electroweak processes at e^+e^- colliders) the program was originally designed for. However, the present version has a much larger range of applicability, especially focusing on beyond the Standard Model (BSM) physics. Not that `WHIZARD` had originally been working with `MadGraph` and `CompHEP` as matrix element generators, while `O'Mega` joined in summer 2001. From version 1.90 on `O'Mega`, which had before been installed externally and its environment variables properly set within `WHIZARD` has been attached to `WHIZARD`. From that version on, `MadGraph` and `CompHEP` are switched off per default and will only be used for compatibility and consistency tests.

From January 2007 on, the new development branch of `WHIZARD` 2.0.0 has been opened and the new version 2.0.0 has finally been released in April 2010. Many design deficiencies of version 1 like event-dependent scales, vast support of hadron collider physics, factorized decays, new event formats like HepMC etc. have been included in the new release. With version 1.94 released in February 2010 we decided to freeze `WHIZARD` version 1, meaning that there will be no new feature developments in the version 1 branch. Further releases will only contain updated documentation, regression and bug fixes. For a description of the features, installation and running of `WHIZARD` 2 confer the abovementioned webpages and the manual and distribution files of `WHIZARD` version 2 directly.

2 Getting Started

While the full details of WHIZARD's capabilities are explained in the later chapters, a simple working example should give some flavor of the program.

2.1 Evaluating a total cross section

First of all, WHIZARD has to be installed on your computer. It is designed to run on UNIX systems and has been successfully tested on several Linux OS as well as MAC OS X. While many features it uses are standard nowadays (see Sec.3), some are not, and you should check first:

- For the O'Mega matrix element generator you need the Objective Caml (O'Caml) language and compiler. See p.??.
- You must have a Fortran compiler which supports the Fortran 95 standard.[1]

WHIZARD has been extensively tested only on Linux PC and Alpha (DEC/Compaq) systems; feedback for other platforms is welcome.

When these conditions are met, you should switch to an empty directory, e.g. whizard, and unpack the source there:

```
cd whizard
tar xzf whizard-1.xx.tgz
```

Next, the package has to be configured:

```
./configure
```

Inspect the output of configure and look for warnings and errors. It may happen that some environment variables have to be set so that the configure script can find all it is looking for (see Sec. 3.3.2). (Except for very old systems, WHIZARD compiles with the system Fortran 95 compiler, and on most operating system, O'Caml is also available as a package.)

After configuration, you have to choose the physics model and the processes you are interested in. You are free to specify an arbitrary list of scattering processes, which you write into the configuration file conf/whizard.prc. The file conf/whizard.prc.default is intended as a guideline: the processes listed there are just examples.

After this is done, the compilation procedure (see below) results in an executable named whizard. This will be an integration program and Monte-Carlo event generator for exactly those processes you have specified. So, in principle, there can be as many different WHIZARD versions as there are possible process lists.

[1] WHIZARD has been checked to work with recent versions of the g95, NAG, Lahey/Fujitsu, Portland (PGF) and Intel compilers (Intel/Linux) and the Compaq compiler (Alpha architecture). If you experience problems, first check the compiler version, since some compiler bugs may have been fixed only very recently. For more information about compilers confer http://projects.hepforge.org/whizard/compilers.html

To be specific, let us consider W pair production with a four-fermion final state:

$$e^+e^- \to \mu^-\bar{\nu}_\mu u\bar{d} \tag{1}$$

The corresponding configuration file `conf/whizard.prc` reads

```
# The selected model
model   SM

# Processes
# Tag           In      Out             Method  Option
#=================================================
cc10            e1,E1   e2,N2,u,D       omega
```

The lines beginning with `#` signs are comments, so the only relevant information are the model definition (the Standard Model in this case) and the lines defining the processes. In the example, there is only one process definition. The particle names follow the `CompHEP` convention, but other conventions may be adopted as well. (You can find all definitions of particle properties and names, as well as the physics parameters of the selected model, in the corresponding model file located in the `conf/models` subdirectory. In the present case, this is `SM.mdl`.)

The keyword `omega` selects the `O'Mega` matrix element generator. Again, the current `WHIZARD` version comes in principle with three different matrix element generators one can choose from (`O'Mega`, `MadGraph`, and `CompHEP`). But note that the latter two would have to be enabled with corresponding `configure` options to be available.

Next, the following command makes the executable and installs it in the subdirectory named `results`:

```
make prg install
```

This runs fully automatically, but it will take some time: The `O'Mega` matrix element generator has to be compiled, then the process matrix element has to be constructed as a Fortran program and compiled, auxiliary programs have to be compiled as well, and so on. Finally, everything is linked into the single program named `whizard`.

The subdirectory `results` is self-contained: Along with the executable, the `make` scripts put there all necessary input files, so the contents of this directory may be moved anywhere else without affecting the functionality of the program.

The `whizard` executable is now a Monte-Carlo generator for the process (1). To run it, change into the `results` directory

```
cd results
```

and edit the input file `whizard.in` there. This file consists of several *input blocks*, each one beginning with a `&` keyword and terminated by a slash.[2] The process ID (`"cc10"` in our case) and the total energy are specified in the first input block. Apart from this, we set the muon mass to zero[3] but leave everything else empty for the moment:

[2] This is the NAMELIST format as defined by the Fortran language standard.

[3] Otherwise, the matrix element would contain the unphysical decay $\mu^+ \to \bar{\nu}_\mu u\bar{d}$ since the u and d quark masses are always zero.

```
&process_input
process_id = "cc10"
sqrts = 500
/

&integration_input /
&simulation_input /
&diagnostics_input /
&parameter_input
Mmu = 0
/
&beam_input /
&beam_input /
```

When this is done, we can start the program:

```
./whizard
```

On screen, WHIZARD will first display a header

```
! WHIZARD 1.97 (May 31 2011)
! Reading process data from file whizard.in
! Wrote whizard.out
!
! Process cc10:
!    e a-e ->  mu a-nu_mu   u a-d
!   32  16 ->   1       2   4   8
```

This states that, at the very beginning, an output file `whizard.out` is written. This file contains the full list of input parameters you have specified along with the default values chosen for the other parameters. Next, the process is listed. The codes below the particle names are binary codes which will become useful when cuts for kinematical variables are specified.

There follow several informational lines which you may ignore since the program takes care of all necessary steps by itself. Nevertheless, some explanation might be in order:

Before starting integration, WHIZARD constructs a phase-space setup. The program uses an adaptive multi-channel integration method. This involves the simultaneous use of multiple phase-space parameterizations (*channels*), which roughly correspond to dominant Feynman diagrams of the selected process. (Note that the absence of a channel only means that it is not taken into account for a specific phase space mapping, but the corresponding Feynman diagrams are nevertheless contained in the evaluated O'Mega amplitude.) Therefore, the physics model is taken into account:

```
! Reading vertices from file whizard.mdl ...
! Model file:       54 trilinear vertices found.
! Model file:       54 vertices usable for phase space setup.
```

The model file `whizard.mdl` is a copy of the original `SM.mdl` before; it contains a list of all vertices relevant for constructing phase space integration channels. Next, these channels are constructed:

```
! Generating phase space channels for process cc10...
! Phase space:       8 phase space channels generated.
! Scanning phase space channels for equivalences ...
! Phase space:       8 equivalence relations found.
! Note: This cross section may be infinite without cuts.
! Wrote default cut configuration file whizard.cc10.cut0
! Wrote phase space configurations to file whizard.phx
```

The resulting set of channels is written to a file `whizard.phx` which is in plain ASCII format. Furthermore, when constructing the phase space the program has found some indication that cuts might be needed, because the final state contains two massless quark jets. (Strictly speaking, in the present case of unequal quark flavors, this is unnecessary, but it is a standard choice.) The corresponding default cut on the dijet invariant mass is written into the file `whizard.cc10.cut0` for informational purposes, and it will be used in the integration.

The weights of the channels will be adapted iteratively. Furthermore, each integration dimension is binned, and the location of bins (the grid) will also be iteratively adapted for each channel. In the present case, this amounts to

```
! Created grids:      8 channels,  8 dimensions with 20 bins
```

After these preliminaries, the program starts integration. It chooses 20,000 calls for each iteration of the adaptive iteration procedure, which takes some time to evaluate. Finally, the output looks like this:

```
! WHIZARD run for process cc10:
!=============================================================================
! It      Calls  Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!-----------------------------------------------------------------------------
! Reading cut configuration data from file whizard.cut1
! No cut data found for process cc10
! Using default cuts.
cut M of   12      within  1.00000E+01  1.00000E+99
! Preparing (fixed weights):   1 sample of      20000 calls ...
   1      20000  2.8867607E+02  1.17E+01    4.05    5.73*  1.59    0.00   1
!-----------------------------------------------------------------------------
! Adapting (variable wgts.):  10 samples of      20000 calls ...
   2      20000  2.7061501E+02  1.03E+01    3.82    5.40*  1.69
   3      20000  2.7528866E+02  1.99E+00    0.72    1.02*  6.88
   4      20000  2.7296493E+02  1.55E+00    0.57    0.80* 12.27
   5      20000  2.7147754E+02  1.34E+00    0.49    0.70* 16.73
   6      20000  2.7038028E+02  1.29E+00    0.48    0.67* 16.04
   7      20000  2.7298452E+02  1.27E+00    0.46    0.66* 14.17
   8      20000  2.7065951E+02  1.26E+00    0.47    0.66  14.94
```

```
   9      20000  2.7150651E+02  1.25E+00    0.46    0.65* 14.46
  10      20000  2.6996907E+02  1.28E+00    0.47    0.67  11.30
  11      20000  2.6978351E+02  1.26E+00    0.47    0.66  13.35
!-------------------------------------------------------------------------------
! Integrating (fixed wgts.):   3 samples of      20000 calls ...
  12      60000  2.7082438E+02  7.23E-01    0.27    0.65  12.12    0.26   3
!-------------------------------------------------------------------------------
```

At the beginning of each integration pass, the kinematical cuts are read, which are either specified by the user in a cut definition file or, if this information is absent as in the present case, the default set of cuts is used. (The letter `M` stands for the invariant mass, while the code 12 stands for the quark pair – add the binary codes 4 [u] and 8 [$\bar{d}$] to get 12.)

The next lines allow the user to follow the convergence of the integral during the adaptation procedure. For each iteration, the display shows the iteration index, the number of phase space points sampled, the total cross section estimate, and the error estimate: the absolute value, the relative percentage, and the relative error multiplied by the square root of the number of calls. The latter number should be of order one; in our example it is slightly below one in the final iterations.

During this procedure, the currently best result is always marked by an asterisk. Note that each integral estimate is independent of each other; no accumulated integrals are shown. In some iterations the accuracy apparently does not improve, but after 12 iterations the grid and weight adaptation has nevertheless reduced the error estimate for a single sample by a factor 5. The last column shows an estimate for the reweighting efficiency when generating actual Monte-Carlo events. Since we have not yet enabled event generation, it is for informational purpose only. Note that this number tends to be less stable than the integration error, but adapting the grids has nevertheless considerably improved this value.

The final result in the last row is evaluated by averaging three further iterations of 20,000 events each. The preceding results are discarded since (at early stages of the adaptation) they may be systematically off the correct result. The total cross section is thus estimated as

$$\sigma_{\mathrm{tot}} = 270.82(72)\ \mathrm{fb} \tag{2}$$

There is also a χ^2 value given which results from comparing the three independent samplings which make up the final integration step. If this value is much larger than 1, we should mistrust the error estimate. Here, the value 0.26 indicates a well-behaved result.

There are a few more lines in the output:

```
!
! Time estimate for generating 10000 unweighted events:    0h 00m 04s
!===============================================================================
! Summary (all processes):
!-------------------------------------------------------------------------------
! Process ID     Integral[fb]  Error[fb]   Err[%]        Frac[%]
!-------------------------------------------------------------------------------
  cc10           2.7082438E+02  7.23E-01    0.27         100.00
```

```
!-------------------------------------------------------------------------
  sum            2.7082438E+02  7.23E-01    0.27         100.00
!=========================================================================
! Wrote whizard.out
! Integration complete.
! No event generation requested
```

The time estimate for event generation (see below) may be taken as a rough guideline, based on the previous reweighting efficiency estimate. Then, the result is repeated – if we had selected more than one process for integration, the individual cross sections would be summarized here. Finally, we get reminded that the input and output data have been saved in the file `whizard.out`. Actually, this file contains only the summary. The process-specific data are collected in the process-specific output file `whizard.cc10.out`.

2.2 Generating events

The results we have obtained can be used for event generation. Apart from the total cross section which is displayed on screen, the adaptive integration procedure produces a set of sampling grids, tailored to the given process and parameter set, which are written on disk and allow the generation of true Monte-Carlo events.

To this end, we restart the program (of course, we could have done it in a single step). We do not like to repeat the lengthy adaptation procedure, therefore we tell `WHIZARD` to reuse the grids. Furthermore, we specify a certain luminosity, say, 10 fb^{-1}. The modified input file `whizard.in` looks like

```
&process_input
process_id = "cc10"
sqrts = 500
luminosity = 10
/

&integration_input
read_grids = T
/
&simulation_input /
&diagnostics_input /
&parameter_input
Mmu = 0
/
&beam_input /
&beam_input /
```

The output looks as before (recall that the results are now read from file), but after displaying the integral the program continues and generates the appropriate number of Monte-Carlo events:

```
! Reading analysis configuration data from file whizard.cut5
! No analysis data found for process cc10
! Event sample corresponds to luminosity [fb-1] =   9.999
! Event sample corresponds to       22344  weighted events
! Generating       2708 unweighted events ...
```

When this is finished, the program displays a summary:

```
! Event generation finished.
!=============================================================================
! Analysis results for process cc10:
! It      Events Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!-----------------------------------------------------------------------------
  13       2708  2.7082438E+02  5.20E+00    1.92    1.00 100.00
!-----------------------------------------------------------------------------
! Warning: Excess events:    5.1       (    0.19% )  | Maximal weight:  1.95
```

The integral is unchanged (by definition), but the given error is now the statistical error which corresponds to the size of the event sample (2708 events). The 100 % efficiency just tells us that there were no additional cuts in the event generation pass; if there were some, it would be the percentage of events that passed the cuts.

The excess events (the sum of all weights exceeding 1) are few enough that they can be safely ignored. Actually, the largest event weight was 1.95 which tells us that the reweighting efficiency estimate – the ratio of the average and the highest weight – has been slightly too optimistic. (In a Monte-Carlo approach of this complexity, it is impossible to calculate the highest possible event weight; it can only be approximated, and there is always a chance for fluctuations.)

2.3 Analyzing events

Clearly, such a run is not very useful without doing anything with the generated events. Fortunately, they have been written to file (in a compressed machine-dependent format) and can be reread in a subsequent run (again, we could also have done it in a single run).

We might like to investigate the distributions of the hadronic invariant mass near the W pole and of the muon energy in our event sample. To achieve this, we set up a file named `whizard.cut5` (the file defining cuts – and histograms – for the 5th pass of the program). It could read

```
# Analysis configuration file for the process ee -> mu nu u d
process cc10
   histogram M of 12 within 70 90 nbin 20
and
   histogram E of  1 within 0 500 nbin 25
```

Again, the binary codes of the particles come into play. Recall that they are in the present example

e^-	e^+	$\to$	μ^-	$\bar{\nu}_\mu$	u	$\bar{d}$
32	16		1	2	4	8

The muon is thus assigned the number 1, and the combination $u\bar{d}$ is assigned the number $12 = 4 + 8$.

We have still to tell the program to reuse the previous event sample. In the input file, we change the block

```
&simulation_input /
```

to

```
&simulation_input
read_events = T
/
```

and restart the program. In the output, the program now says

```
! Found  2 analysis configuration datasets
```

and shows two identical summaries, one for each dataset. However, it has also generated the required distributions. Those are in the file `whizard.cc10.dat` and can be inspected there:

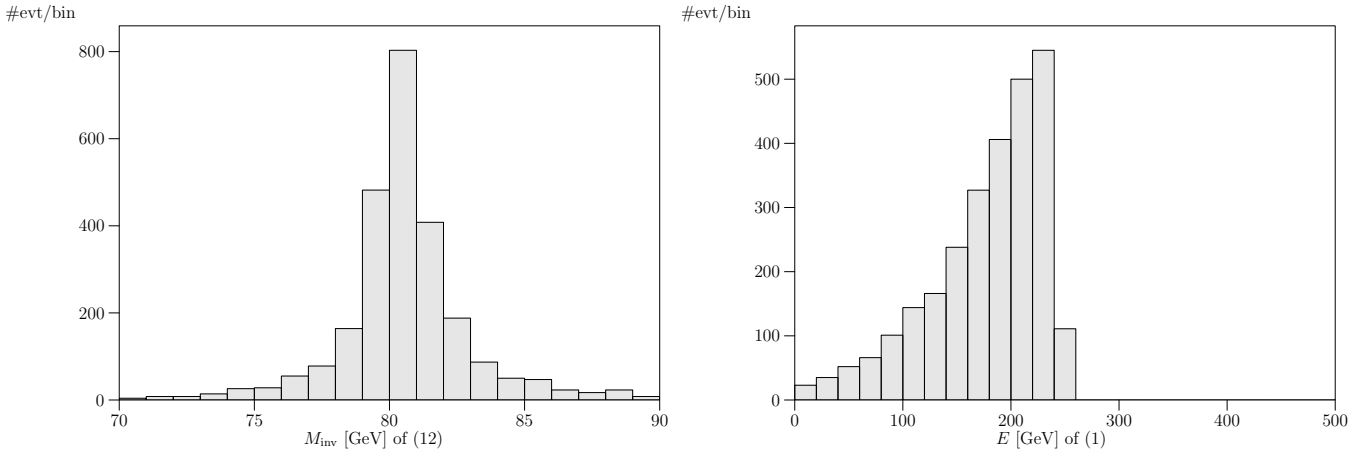
```
!=============================================================================
 ! WHIZARD 1.97 (May 31 2011)
 ! Process cc10:
 !    e a-e ->  mu a-nu_mu   u a-d
 !   32  16 ->   1       2   4   8
 ! Analysis results for process cc10:
 !
 ! Histograms:
 !
 histogram M of  12     within  7.00000E+01  9.00000E+01  nbin   20
   70.5000000        5.00000000        2.23606798        0.00000000
   71.5000000        13.0000000        3.60555128        0.00000000
   72.5000000        10.0000000        3.16227766        0.00000000
   73.5000000        17.0000000        4.12310563        0.00000000
...
```

The first column shows the bin midpoints, the second column the number of events, and the third column the statistical error $\sqrt{n_{\rm evt}}$. The fourth column shows excess events if there are any.

The contents of this file can be displayed graphically by standard analysis packages. For convenience, `WHIZARD` is able to do this by itself, using a macro package for the MetaPost drawing program[4]. The simple command [5]

```
make plots
```

will execute all necessary steps automatically. The result is a PostScript file called `whizard-plots.ps`. For the case at hand, the plots are shown here:



As expected, the hadronic invariant mass concentrates around the W mass, while the muon energy shows a smooth distribution.

2.4 Further options

The above example covers only basic features of `WHIZARD`. Some further possibilities, which are explained in the subsequent chapters of this manual, are:

- Evaluate and sum over arbitrary processes.
- Change physical parameters (masses, couplings)
- Calculate processes in the MSSM (or other BSM models)
- Reweight matrix elements by user-defined functions.
- Insert parton structure functions (hadron collider) from the LHAPDF (or alternatively PDFLIB), beamstrahlung/ISR (e^+e^- collider), or the Weizsäcker-Williams approximation for photons.

[4]The macro package is called `gamelan` and is included in the distribution. MetaPost must be available on the system; this is determined by `configure` when `WHIZARD` is installed.

[5]Doing this the first time, this `make`, or `make gml` has to be issued in the main directory, so `gamelan` is compiled. Later, it also works in the `results` subdirectory where the files reside.

- Choose polarized beams.
- Insert user-defined cuts.
- Analyze the final-state polarization.
- Mix several processes in event generation.
- Fragment the full system including beam remnants using `PYTHIA`.
- Fragment the final-state particles using `JETSET`.
- Write events to file in ASCII or STDHEP format and analyze them by external programs.
- [For historical reasons and cross-checks: use different matrix-element generators (`CompHEP` and `MadGraph` besides `O'Mega`).]

3 Installation

3.1 Sources

WHIZARD is Free Software (under the GPL Gnu Public License) and the sources can be obtained from

```
http://projects.hepforge.org/whizard
```

or alternatively from

```
http://whizard.event-generator.org
```

The command

```
tar xzf whizard-1.xx.tgz
```

will unpack the sources in the current working directory (note that the tarball does not generate a new directory). After unpacking, the following subdirectories will exist:

`doc` contains the manual

`conf` contains the configuration file and default input files

`kinds-src` contains the definition of the Fortran REAL kinds

`omega-src` contains the O'Mega distribution, the default matrix element generator of WHIZARD

`circe-src` contains the CIRCE beamstrahlung library

`circe2-src` contains the CIRCE2 library

`vamp-src` contains the VAMP integration library

`gml-src` contains the gamelan sources which allow a graphical representation of WHIZARD results

`whizard-src` contains the actual WHIZARD sources

`bin` contains the scripts needed for running WHIZARD and, after compilation, the WHIZARD executable

`lib` is created during compilation and holds the resulting program libraries which when linked make up the WHIZARD executable.

include contains after compilation the Fortran include files and files containing the parameters of the model in use

`processes-src` is created during compilation and holds the source code for the generated matrix elements and corresponding Feynman graph pictures

`results` contains the `WHIZARD` executable and input files after installation

`fmf-src` contains the LaTeX macros for drawing Feynman diagrams

`config.scripts` contains maintenance scripts

[`chep-src` contains the `CompHEP` sources]

[`mad-src` contains the `MadGraph` sources]

[`helas-src` contains the `HELAS` library which is used by `MadGraph`]

3.2 Prerequisites

3.2.1 Necessary

The following programs must be available on the host system in order to successfully compile and run `WHIZARD`[6] :

- GNU Make: Required for compiling `WHIZARD`. The scripts in `WHIZARD` also make use of standard UNIX tools such as `grep`, `sed`, etc.
- PERL 5: Matrix element generation is controlled by PERL scripts.
- Fortran 95: The programming language of the actual `WHIZARD` code. Be aware that some Fortran compilers, unfortunately, still have internal bugs that prevent successful compilation of `WHIZARD`. Meanwhile, the `gfortran` shipped with the `gcc` bundle compiles `WHIZARD`from version 4.3 on (which is the oldest compiler version still supported by `gcc`).
- Fortran 77: This is required only for interoperability with pre-compiled Fortran 77 libraries such as the `CERNLIB`. If this is not needed, a symlink to the Fortran 95 compiler should suffice.
- `O'Caml` Objective Caml programming language: this is needed for the compilation of `O'Mega` in a recent version (3.04 or newer). This has become part of of most (if not all) Linux distributions. Otherwise, you can get it from

  `http://pauillac.inria.fr/ocaml/`

3.2.2 Optional

The following programs are not needed for compiling and running `WHIZARD`, but only for some additional features:

[6] A C compiler is needed by `CompHEP`. The compiler and compiler flags are set in `chep-src/unix_com/CC`.

- The LHAPDF library for parton distribution functions. The package can be found here http://projects.hepforge.org/lhapdf. Although it is not mandatory, it is highly recommended to compile LHAPDF with the same Fortran95 compiler as `WHIZARD`, because otherwise you might need to link compiler libraries into the `WHIZARD` binary.

- The STDHEP library for writing events in STDHEP format. This library is contained in the CERN library in pre-compiled version only. (Using it may require the corresponding Fortran77 runtime library, which is determined automatically by `configure`.) Alternatively, if you do not want to rely on the CERNLIB, you could get the STDHEP library from here http://cepa.fnal.gov/psm/stdhep/getStdHep.shtml.

- The PDFLIB library from the CERN library (`CERNLIB`) which accounts for parton distribution functions. (Using it may require the corresponding Fortran77 runtime library, which is determined automatically by `configure`.) Although this is still supported within `WHIZARD` 1, it is considered to be deprecated, and one should use LHAPDF instead.

  When you still want to use this feature, and there is a precompiled version on your system (e.g. compiled with `f77` or `g77`), then you probably need to include `FCFLAGS=-lg2c`. Furthermore, you might need to compile the file `f77_files.f` in the `whizard-src` directory with the command `<your f95 compiler> -c f77_files.f` and include the object file in the following line in the file `bin/whizard.ld`:
  `OBJ="$* processes-src/*.o whizard-src/f77_files.o"`.

- PYTHIA can be used for decaying and hadronizing the events generated by `WHIZARD`. If the `PYTHIA` library is present on the host system, it will be called for that purpose from inside `WHIZARD`, so no external interface is needed. The library is contained in the CERN library in pre-compiled version. (Using it may require the corresponding Fortran77 runtime library, which is determined automatically by `configure`.) **Important:** Note that the `WHIZARD` 1 `configure` setup only supports the `PYTHIA` library as it is contained in the CERNLIB, using a version compiled by yourself might lead to the need to modify several `configure` options or `Makefile`s.

- LaTeX for typesetting the documentation (including this manual) Note that the cleanest and most complete way to get LaTeXis to install the TeXLive bundle.

- HeVeA for making the HTML version of the manual

- MetaPost for on-line generation of histogram and data plots. This is again contained in the TeXLive bundle.

- noweb for weaving the program sources.

- autoconf for creating a configure script in the restricted bundle.

3.2.3 Independent components

WHIZARD contains the following four sub-programs which are basically independent programs by themselves, but have been agreed upon by their authors to be merged into the WHIZARD bundle from version 1.90 on. This has been done in WHIZARD version 1.90 and the 1.9x distribution series by just attaching the code of these tools to the WHIZARD main core, while in WHIZARD 2 the sub-components have been recovered as completely, self-configurable and independently maintainable tools.

The four tools discussed above are

- O'Mega the matrix element generator. This program is now steered also via the WHIZARD HepForge page http://www.hepforge.org/downloads/whizard. Note that only the O'Mega versions up to 0.xx are compatible with WHIZARD 1.xx. The newer versions carrying the same version number as WHIZARD 2.x.x have been designed and modified to work with WHIZARD 2 only. Again, only for WHIZARD up to version 1.51 a separate download of O'Mega is necessary, from 1.90 on the matrix element generator is included in the WHIZARD tarball.

- CIRCE and CIRCE2 (for lepton collider beam spectra and photon collider spectra) are now steered from the WHIZARD HepForge page as well, http://www.hepforge.org/downloads/whizard. They have been re-named there as CIRCE1 and CIRCE2, respectively. CIRCE itself has been included into WHIZARD from the very beginning, while CIRCE2 is part of the distribution since version 1.28. Both programs have been attached with an additional WHIZARD interface, which has been modernized and updated in WHIZARD 2.

- VAMP is the multi-channel adaptive integration routine of WHIZARD which has been part of the WHIZARD core since the very beginning of the program. It is now as well steered via the WHIZARD HepForge page http://www.hepforge.org/downloads/whizard.

3.2.4 Alternative matrix element generators

The following two matrix element generators in modified versions are included in the WHIZARD distribution and need not be obtained separately.

The URLs below should be inspected for the documentation, references, and license conditions:

- CompHEP:

  http://comphep.sinp.msu.ru

  The CompHEP version which is included in WHIZARD is completely out-of-date (version 3.2.18), but has an additional Fortran 95 interface. It is kept mainly for the possibility of generating spin-summed matrix element for simple processes, which the other methods (which use helicity amplitudes) cannot provide as efficiently. Furthermore, with CompHEP the definition of new models is straightforward and does not require programming.

- MadGraph/HELAS:

`http://madgraph.physics.uiuc.edu`

Again, only a very old version from the MadGraph 3 release series with a modified Fortran 95 interface is supported by `WHIZARD`. Especially, the version attached to the `WHIZARD` distribution does not contain any Higgs self couplings.

It should be stressed that the current `CompHEP` and `MadGraph` versions are much more powerful than the versions included within the `WHIZARD` distribution and are full-fledged event generators by themselves.

3.3 Configuration

Running the configure script

```
./configure
```

will check the availability of programs and utilities and generate the Makefiles appropriate for the host system. At the end of the configuration, a summary of the available/enabled and unavailable/disabled features is printed.

If desired, `configure` can be re-run. This is useful if you change environment variables or configure options (see below). If you would like to make the changes active and you have already made or run the `WHIZARD` executable, it may be necessary to do a `make clean` before you redo `make prg` or similar.

If you only wish to re-generate the Makefiles without changing anything else, you can run `config.status`. You may also use the command

```
make distclean
```

in order to remove everything including the files `whizard.prc` and `whizard.in` and thus restore the original status of the package. [7]

3.3.1 Configure options

According to the standard, the available options to `configure` can be viewed by typing

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

However, although this prints a long list of standard options, none of them are relevant except for the ones which enable or disable parts of the `WHIZARD` package. If you wish to disable a module, type

```
./configure --disable-MODULE
```

[7] Note that when you have `noweb` installed on your system, `make distclean` will erase all Fortran90 files in the directory `whizard-src` including `user.f90`. This will lead after re-configuration and doing `make prg install` again to the error message `make[1]: *** No rule to make target 'user.f90',needed by '.depend'. Stop.`. In that case, just copy `user.f90.default` to `user.f90` and repeat `make prg install`.

where MODULE is the tag of one of the modules in the list below. There can be multiple disable options in the command line. Conversely, the command

```
./configure --enable-MODULE
```

will configure WHIZARD with module MODULE enabled.

Here is the list of modules that can be switched on or off in this way:

Module	Tag	Purpose	Enabled by default
O'Mega	omega	Matrix elements	yes
CompHEP	chep	Matrix elements	no
MadGraph	mad	Matrix elements	no
CIRCE	circe	Beamstrahlung	yes
CIRCE2	circe2	Beamstrahlung	yes
LHAPDF	lhapdf	Parton structure functions	yes
PDFLIB	pdflib	Parton structure functions	yes
PYTHIA	pythia	Fragmentation and hadronization	yes
STDHEP	stdhep	Binary event files for further processing	yes
MetaPost	metapost	PostScript Histograms	yes

There are a few more configure options that might be relevant, but are not sub-packages to be disabled or enabled. The most interesting option for the user, `--enable-quadruple` allows to use quadruple precision, if the compiler supports this (like e.g. the standard `gfortram`). As a default it is switched off. Note that in some earlier WHIZARD versions this was set by means of an environment flag (cf. next subsection), but has been changed into a configure option lately. The option `--enable-fc-profiling` is only intended for profiling monitoring for developpers, while the option —`--enable-fc-impure` would trigger a compilation of the O'Mega libraries with non-pure and non-elemental functions. This allows to use `print` statements in the matrix elements for debugging.

3.3.2 Configure environment

The following environment variables are recognized by `configure`. They need to be set only if `configure` cannot find a program or library, or if a choice different from the default one is desired. Path and program names should be specified absolute, not relative to the working directory.

GMAKE: GNU Make.

FC: The Fortran 90/95 compiler. (The aliases **F95**,**F90** also work.)

FCFLAGS: Flags to be passed to the Fortran 90/95 compiler. (The aliases **F95FLAGS**, **F90FLAGS** also work.)

F77: The Fortran 77 compiler. This is needed for correctly linking precompiled Fortran 77 libraries (see below). It is highly recommended to use whenever possible the same compiler for the Fortran 95 and Fortran 77 code.

FFLAGS: Flags to be passed to the Fortran 77 compiler.

PERL: The PERL executable.

LIBS: Libraries to pass to the linker, e.g. `-l<library>`.

USERLIBS: To link additional libraries before the default ones.

SYSTEMLIBS: To link additional libraries after the default ones.

LDFLAGS: linker flags, e.g. `-L<lib dir>` if you have libraries in a nonstandard directory `<lib dir>`.

TMPDIR: Directory where temporary files are stored. By default, `/tmp` is used.

LHAPDF_DIR: The directory where the LHAPDF library resides. We strongly advice to compile LHAPDF with the same Fortran 90/95 compiler used for the compilation of `WHIZARD`.

CERNLIB_DIR: The directory where the precompiled (F77) CERN libraries reside (optional). This includes the `STDHEP`, `PDFLIB` and `PYTHIA` libraries together with auxiliary libraries. If necessary, individual environment variables STDHEP_DIR, PDFLIB_DIR and PYTHIA_DIR can be used to override the setting of CERNLIB_DIR. If CERNLIB_DIR is not set, `/cern/pro/lib` is assumed.

NOWEB_DIR: Directory where the `notangle`, `noweave`, `cpif` executables reside (optional). With these programs the Fortran 95 sources and a documented program listing can be (re-)generated from the ultimate source `whizard.nw`.

GTAR: GNU tar (optional).

Instead of specifying configure options and environment variables on the command line, they may be defined in the script `config.site` which is executed each time configure is run.

There are a few more environment variables like `CC`, `CFLAGS`, `CPP`, `CPPFLAGS` concerning the `C` compiler, precompiler and their corresponding flags. As this is only relevant for compiling `CompHEP` we do not mention these flags here any further[8].

3.3.3 Test runs

After configuration is complete, one can test if the whole system works by executing

```
make test
```

[8] There is also CHEP_MAXDIAG, the maximum number of diagrams contained in one `CompHEP`-generated file. If there are more diagrams, files will be split. A value too large may lead to memory overflow for particular compilers. Default value: 100.

This will set up a Standard Model standard candle process using all activated matrix element generators, i.e. per default `O'Mega`, compile the whole package and start the program for the process $e^+e^- \to \bar{\nu}_e\nu_e H$ in the Standard Model (in up to three versions of the same process, but compiled by different generators if `CompHEP` or `MadGraph` are enabled.). With the parameter set taken from the default input file, this should result in equal cross sections for all activated matrix elements, about 85 fb each.[9] Finally, a small sample of 100 events is generated and, if the `PYTHIA` module is activated, fragmented by `JETSET`.

While the above test is more like a demo, there are actually self-tests provided that check whether the results obtained with the three different matrix-element generators actually agree. Currently, the possibilities are

```
make test-QED
make test-QCD
make test-SM
make test-MSSM
```

Note that the selftests compare the string expressions of the program output for the three different matrix element generators. As the numerical routines are slightly different, the results may deviate from each other and hence produce a fake failure of the self test. Just have a look at the numbers. Only the SM test should report a few real disagreements. These come from the fact that currently O'Mega does not couple the Higgs to muons or strange quarks, while Madgraph does not treat Higgs pairs correctly. The MSSM test currently has just `O'Mega` matrix elements, since the built-in `CompHEP` and `MadGraph` versions do not support the MSSM. However, the results can be compared with the numbers in the appendix of [12].

3.3.4 Cleanup

Unnecessary intermediate files will be removed if you say

```
make clean
```

If the process configuration file is modified to hold a new list of processes for the same or a different physics model, new `make` calls will remake the system as needed, so under normal circumstances no files have to be deleted explicitly. If you are not sure about the current status, you may remove the process-dependent files by

```
make proclean
```

To do a thorough cleanup, type

```
make realclean
```

This will also remove the `CompHEP` and `MadGraph` executables used for generating matrix elements, recovering the state just after configuration.

Finally,

[9] If, for the selected Fortran compiler, command-line arguments are not supported, the program will just display the list of processes and exit.

```
make distclean
```

removes also the files resulting from configuration, thus recovering the initial state of the package as it comes in the distribution. Note that this also deletes some files which you may have edited yourself: `whizard.prc` and `whizard.in` in the `conf` directory, the input files in the `results` directory, and the file `user.f90` in the `whizard-src` directory, so save them elsewhere if you do not want to lose their contents. After `make distclean` has been executed, you have to run `configure` before any further `make` command can be executed.

For further options, see the comments in the header of the master Makefile.

4 Running WHIZARD

4.1 Setting up the process list

WHIZARD is able to cope with arbitrary scattering processes allowed by the selected physical model. The program is not a library of processes, but the user can compile a list of processes he is interested in, write a file whizard.prc and put it in the subdirectory conf. If no file is provided by the user, the file whizard.prc.default (Fig. 1) will be used. Note that as per default CompHEP and MadGraph are disabled all processes by these generators will be skipped. Only O'Mega and trivial test matrix elements will be produced. The user may take this file as a template, insert or delete processes as desired, and save the new version as whizard.prc.[10]

4.1.1 Model selection

WHIZARD is not restricted to the Standard Model. The physics model is selected in the header of whizard.prc, respectively.

For each model, all particles, parameters, and the vertices which go into the phase space setup are defined in a single file with extension .mdl. The vertex definitions include the complete specification of Feynman rules for MadGraph and CompHEP, while the O'Mega vertices are currently hard-coded in the corresponding executable. The model files can be found in the conf/models subdirectory. Along with the master files SM.mdl, MSSM.mdl, etc., for each model XXX there is also a file parameters.XXX.omega.f90 which contains the translation to the parameter definitions used internally by O'Mega.

The distribution contains the following models (see Tab. 2):

QED: QED with three lepton generations.

QCD: QCD with three quark generations.

SM: The Standard Model with e, $\sin\theta_W$ and M_Z as independent parameters in the electroweak sector. Some remarks are in order[11]:

- The u and d masses are zero, all other masses are non-zero by default and user-definable.
- All fermions, except for the top quark, have zero width.
- By default, the CKM matrix is the unit matrix, but for each model where it matters, a variant with nontrivial CKM matrix is available. Keep in mind that keeping the full CKM matrix may result in less efficient calculations.

[10] A better method is to give the user-created file a different name and use a symbolic link to whizard.prc instead. The same applies to whizard.in.

[11] The MadGraph version implemented in WHIZARD does not treat multi-scalar vertices correctly. Don't use it if you are interested in double (or triple) Higgs production. CompHEP uses Feynman gauge (i.e., includes ghosts as external states) for a more efficient evaluation of electroweak processes, while the other generators use unitarity gauge. This should not be visible to the user. Note also that CompHEP provides polarization-averaged matrix elements only. There is a CompHEP model using unitarity gauge, SM_ug, which turns out to be less efficient in many cases).

```
# WHIZARD configuration file

# The selected model
model   SM

# Processes
#  Methods: chep=CompHEP, mad=MadGraph, omega=O'Mega, test=trivial)
#  Options: s      selected diagrams (CompHEP/MadGraph)
#           r      restricted intermediate state (O'Mega)
#           c      apply exact color algebra (O'Mega)
#           n:XXX  coupling order (MadGraph)
#           w:XXX  width scheme (O'Mega)
#           p      transfer polarization (test)
#           u      unit matrix element (test)
#
# Tag           In      Out             Method  Option
#=====================================================
ee_c            e1,E1   e1,E1           chep
ee_m            e1,E1   e1,E1           mad
ee_o            e1,E1   e1,E1           omega
ww_c            e1,E1   W+,W-           chep
ww_m            e1,E1   W+,W-           mad
ww_o            e1,E1   W+,W-           omega
zh_c            e1,E1   Z,H             chep
zh_m            e1,E1   Z,H             mad
zh_o            e1,E1   Z,H             omega
nnh_c           e1,E1   n1,N1,H         chep
nnh_m           e1,E1   n1,N1,H         mad
nnh_o           e1,E1   n1,N1,H         omega
nnbb_m          e1,E1   n1,N1,b,B       mad
nnbb_o          e1,E1   n1,N1,b,B       omega
```

Figure 1: *Default process configuration file. Only `O'Mega` processes will be generated per default, the others will be skipped.*

  - The Higgs boson is coupled to all massive fermions, with one exception: `O'Mega` has no Higgs couplings to the strange quark.

`SM_ac`: The Standard Model with anomalous couplings. This is usually considered in the context of dynamical electroweak symmetry breaking, hence, by default, the Higgs mass is set to a large value. `O'Mega` supports anomalous trilinear and quartic gauge couplings[12].

`SM_km`: This model has been included in version 1.90. It is intended for LHC or CLIC physics of Higgsless models or models without unitarizing UV completion. A K-matrix unitarization guarantees a physically sane description for LHC and CLIC energies. More details can

[12]The `CompHEP` version currently has only the anomalous quartic couplings. The `MadGraph` version is just the Standard Model.

be found in [13].

`MSSM`: The Minimal Supersymmetric Standard Model (MSSM)[13]. The input data may be provided in the SUSY Les Houches Accord format (see Sec. 4.4.7). Typically, such a file is generated by automatic SUSY spectrum and decay packages.

For convenience, there are two MSSM input files provided in the `conf` directory: `sps1a.in` implements the mass spectrum of the standard SUSY point SPS1a [14] with zero particle widths; this file has been used for generating the comparison results of Ref. [12]. The file `sps1ap.in` corresponds to the reference point SPS1a' as it has been defined for the SPA convention [15]; this file includes width values for all particles that are computed, in some cases, with higher-order corrections and running couplings incorporated. Be aware that, for a consistent tree-level calculation, it may be necessary to compute widths in a different convention. This caveat is discussed further in Ref. [12].

`MSSM_Grav`: This variant of the MSSM has been included in version 1.93. It contains the MSSM with an additional gravitino and is intended for the studies of GMSB models.

`NMSSM`: This is the Next-to-Minimal Supersymmetric SM, which has been contributed to `WHIZARD` by Felix Braam [16].

`PSSSM`: A non-minimal MSSM version motivated by an $E_6 \to$ Pati-Salam GUT group. This is a very extended model which has been implemented by Daniel Wiesler.

There are more BSM models, whose inclusion into `WHIZARD` started from version 1.51 on:

`Littlest`: The Littlest Higgs model, which contains in addition to the SM a B' (or γ'), a W'/Z' triplet, a complex isospin-2 scalar multiplet ψ, and a t'.

`Littlest_Eta`: The variant of the Littlest Higgs model which contains a pseudoscalar η instead of the B' vector boson.

`Simplest`: A different Little Higgs model which has recurrences of all SM fermions and new isodoublet gauge bosons.

`Simplest_univ`: A variant of the previous model with universal, but not anomaly-free, fermion assignments.

`Xdim`: The SM augmented by a spin-2 graviton as a toy model

`GravTest`: The SM augmented by a photino and a gravitino as a toy model.

`UED`: Universal Extra Dimensions.

`Zprime`: The SM with a completely generic Z' resonance.

[13]The MSSM is supported only by the `O'Mega` matrix element generator, while the included `MadGraph` and `CompHEP` versions will fall back to the Standard Model.

Model type	with CKM matrix	trivial CKM
QED with e, μ, τ, γ	–	`QED`
QCD with d, u, s, c, b, t, g	–	`QCD`
Standard Model	`SM_CKM`	`SM`
SM with anomalous couplings	–	`SM_ac`
SM with K matrix	–	`SM_km`
MSSM	`MSSM_CKM`	`MSSM`
MSSM with gravitino	–	`MSSM_Grav`
NMSSM	`NMSSM_CKM`	`NMSSM`
E_6 SSM	–	`PSSSM`
Littlest Higgs model	–	`Littlest`
Littlest Higgs model (ungauged $U(1)$)	–	`Littlest_Eta`
Simplest Little Higgs model	–	`Simplest`
Simplest Little Higgs (universal coupl.)	–	`Simplest_univ`
UED	–	`UED`
SM with Z'	–	`Zprime`
SM with graviton resonance	–	`Xdim`
SUSY toy model with gravitino	–	`GravTest`
SM as template	–	`Template`
SM, all unitary gauge, `CompHEP` model	–	`SM_ug`

Figure 2: *Summary of models currently supported by* ***WHIZARD****.*

`Template`: This is just a copy of the SM. It might serve as a template if the user wants to implement his/her own model.

The user might wonder whether this list can be extended. Actually, it is trivial to set up, e.g., a SM version with a different input parameter scheme or different particle naming conventions, etc.: Copy `SM.mdl` to, e.g., `SM_new.mdl`, make the changes in that file, and provide copies of the corresponding files for use of the matrix element generators[14]. There is a model `Template` as described above, which is just the SM and might serve as a template to implement the user's own model. However, for this task one needs to modify the module `Template` in `omega-src/bundle/src/models5.ml` (which requires programming in `O'Caml`) and then the corresponding files `Template.mdl` and `parameters.Template.omega.f90` in `conf/models`. This is a rather cumbersome task. There is fortunately an easier way: using the interface of `WHIZARD` to the program `FeynRules` [17]. Here, one can write down a Mathematica file containing a Lagrangian in a very intuitive language, and then `FeynRules` derives the mass eigenstates, mixing and Feynman rules and imports the model into `WHIZARD`. Although this has been developped already for `WHIZARD` version 1 and is fully supported and functional for that

[14]Furthermore, by writing the Feynman rules into a Lagrangian file, one may easily modify or set up a new model for `CompHEP`, which then can be used by `WHIZARD`. Modifying the model for `O'Mega` requires changing program sources, however. The same holds for `MadGraph`, if you are interested in Lorentz or Dirac structures that are not present in the SM.

version, we refer to the manual of version 2 for more information, or alternatively the WHIZARD Wiki page http://projects.hepforge.org/whizard/trac/wiki.

4.1.2 Process list

WHIZARD is capable of computing $2 \to n$ scattering processes and $1 \to n$ decay processes. In the former case, the overall result (the integral) displayed by the program will be the total cross section in fb. In the latter case, it will be the partical decay width in GeV.

For scattering processes, $n = 1$ (single-particle production) is allowed[15][16]. Obviously, this requires at least one beam to be structured, such that a energy spectrum is available for the initial state.

Each process is defined by a single line in `whizard.prc`, for instance:

```
h_production    e1,E1   n1,N1,H         omega
z_decay         Z       e1,E1           omega
z_production    e1,E1   Z               omega
```

(see Fig. 1). The meaning of the entries within a line is as follows:

The **first entry** is a unique alphanumeric tag which will be used to identify the process. It can be arbitrarily chosen, but should be a valid Fortran 95 token, lower case only. In particular, one may use lower-case characters and digits as well as the underscore, where the first character is not a digit.

The **second entry** defines the (partonic) initial state. It consists of two particle names, separated by a comma, with no space in between. The particle names are those given in the model file mentioned above, where typically for each particle several possible names are listed. As an example, in Fig. 3, the Standard Model particles are displayed as they appear in `SM.mdl`. This file also contains alternative names, quantum number assignments, and the physical parameters of the model.

The **third entry** defines the final state. It consists of one, two or more particle names, separated by commas, with no space in between.

Flavor sums[17] are defined by particle names separated by colons. For instance, the line

```
eeqq  e1,E1   u:d:s,U:D:S     omega
```

will create a sum of all processes $e^-e^+ \to q\bar{q}$, with q being any of the light quarks. Similarly,

```
qqmm  u:d:s,U:D:S   e2,E2     omega
```

[15] It does not work for CompHEP matrix elements, however.

[16] The current implementation is not well suited for single-particle processes if the initial-state energy distributions are strongly peaked. A similar caveat holds for processes with an s-channel resonance, for instance, $e^+e^- \to \mu^+\mu^-$ with a continuous beamstrahlung/ISR spectrum for the incoming electrons, integrated over the whole energy range. Nevertheless, with large statistics reasonable results can be obtained. Some improvement has been achieved by using a specific mapping for s-channel like topologies.

[17] Not available for CompHEP or MadGraph matrix elements)

Particle	Name	Antiparticle	Mass	Width
d-quark	d	D		
u-quark	u	U		
s-quark	s	S	ms	
c-quark	c	C	mc	
b-quark	b	B	mb	
t-quark	t	T	mtop	wtop
electron	e1	E1	me	
muon	e2	E2	mmu	
tau-lepton	e3	E3	mtau	
e-neutrino	n1	N1		
μ-neutrino	n2	N2		
τ-neutrino	n3	N3		
gluon	G			
photon	A			
Z-boson	Z		mZ	wZ
W-boson	W+	W-	mW	wW
Higgs	H		mH	wH

Figure 3: *Particle names (**CompHEP** naming scheme) and parameters for the Standard Model.*

creates a sum of all processes $q\bar{q} \to \mu^-\mu^+$. This is especially suitable for hadronic collisions. One should note that the mirror process $\bar{q}q \to \mu^-\mu^+$ is *not* automatically included. To get the correct results for a hadron collider, one can either double cross section values and symmetrize final-state distributions[18] (for a symmetric initial state, e.g. pp) or define the process as[19]

```
qqmm u:d:s:U:D:S,u:d:s:U:D:S e2,E2 omega
```

Note that in the diagnostics messages at runtime, only a single flavor assignment will be shown, which is used as a template for generating phase space parameterizations. For the matrix element evaluation and event generation, however, all valid flavor assignments are used.

To make this more readable, there is the possibility to specify aliases: The above definition is equivalent to

```
alias  q  u:d:s:U:D:S
qqmm   q,q   e2,E2    omega
```

[18]Naively summing over identical particles in the final state may result in double-counting, and becomes very inefficient if many combinations are allowed. The 'c' option takes care of both issues (see below).

[19]If the list of flavor sums or the number of particles in the process becomes very long, it may happen that the compiler complains for the number of continuation lines becoming too large. This problem is easily fixed by increasing the number of continuation lines accepted by the compiler. You should consult your Fortran compiler documentation for the appropriate flag which may then be set in the `config.site` file. Don't forget to clean up and configure before restarting the compilation.

Alias definitions can be arbitrarily mixed with process definitions. They are applied consecutively to all in- and out-state specifications below their definition.

The particles to be summed over must have identical spin, mass and color representation; otherwise the summation will not work. This feature is therefore useful mainly for summing over fermions with identical quantum numbers (quarks or leptons).

The order of particles in the final state can matter if PYTHIA/JETSET is used for fragmenting events. If no information on color connections is available in simulated events (for O'Mega matrix elements if the 'c' option is not set [20]), the (dominant) color configuration has to be inferred from the particle ordering, if at all possible [21]. To transfer the color connections to JETSET, particles that make up a color-singlet string should be put together, beginning with the quark, possibly intermediate gluons, and ending with the antiquark. This suffices for well-defined flavor states. If flavor sums are used and more than one quark pair is present, all possible orders of particles will show up in the final state.

The **fourth entry** selects the method for the generation of the matrix element, while the **fifth** entry is either absent or consists of a string of code letters separated by commas (blanks in between are allowed). Some code letters are followed by a colon ':' that indicates a value. As a default there are two options for matrix elements, either the method omega using the intrinsic generator O'Mega or the method test which produces dummy matrix elements for certain consistency tests described below.

For cross checks and debugging there are also (when enabled during the configuration of WHIZARD) the methods chep and mad for using matrix elements generated by CompHEP or MadGraph, respectively. These features have become obsolete for the user at latest with version 1.50 or 1.90, when O'Mega become the main or default generator WHIZARD. For completeness reasons the details and options for these two generators will be described at the end of this subsection.

O'Mega can generate matrix elements with an arbitrary number of final-state particles, limited only by the host resources (WHIZARD has extensively been tested only for processes up to $2 \to 6$, some very special cases of $2 \to 7$ or $2 \to 8$ have been studied). For multiparticle processes, code generated by O'Mega is expected to be the fastest, since more redundancies are eliminated from the matrix element evaluation.

There are several options (entries in the fifth column) for the O'Mega matrix element generator:

r: The letter r indicates restrictions on intermediate states for the amplitude. Effectively, this also selects classes of Feynman diagrams. Example:

```
nnh         e1,E1   n1,N1,H   omega
nnh_zh      e1,E1   n1,N1,H   omega  r: 3+4~Z
```

For more details, see Sec. 5.2.

[20] And also for CompHEP.

[21] CompHEP, unfortunately, reorders particles according its own conventions.

`w`: The letter `w` modifies the width treatment for unstable particles. The default behavior is the expected one: the width is inserted only for s-channel propagators. This may lead to gauge-invariance issues. The option `w:fudged` (or abbreviated `w:f`) applies the 'fudge-factor' prescription for unstable particles where all terms are collected over the respective propagator denominators (in contrast to the usual fixed-width prescription). This prescription is useful if the process includes the exchange of photons and W bosons in the t-channel, for instance:

```
enw         e1,E1  n1,E1,W-    omega    w:fudged
```

A fixed-width option can be enforced by specifying `w:constant` (or `w:c`). Finally, all widths can be forced to zero both in the s- and t-channel by the option `w:zero` (or `w:z`).

- `O'Mega` comes in two versions: By default, only the leading-$1/N_c$ color factor is generated, and no color-flow information is available. If the option flag `c` is given, `WHIZARD` uses the tensor-product feature for process definitions of `O'Mega` originally intended for flavor summation to treat color exactly and to generate color-flow information. Example:

```
uudd_leading   u,U  d,D      omega
uudd_full      u,U  d,D      omega   c
```

The modified version also eliminates some redundancies in summing over flavor.

In the current implementation, the default (leading-color) version can be more efficient (unless many flavor states need to be summed over), so it is still useful if color is not an issue. This is the reason why we decided to keep the leading-color version as the default. Note that the `O'Mega` generator within `WHIZARD` 2 takes care of the color-flow information completely by itself and delivers matrix elements that contain the full color correlations as a default.

The `test` method generates a constant matrix element, independent of the particles supplied for the process. There are two choices for the normalization:

1. If the option 'u' is given, the *matrix element* is unity. With zero masses and no cuts, the integral should be exactly

$$\sigma_n(s) = \frac{(2\pi)^4}{2s}\Phi_n(s) = \frac{1}{16\pi s}\,\frac{\Phi_n(s)}{\Phi_2(s)}, \tag{3}$$

where the volume of the massless n-particle phase space is given by

$$\Phi_n(s) = \frac{1}{4(2\pi)^5}\left(\frac{s}{16\pi^2}\right)^{n-2}\frac{1}{(n-1)!(n-2)!}. \tag{4}$$

For $n \neq 2$ the phase space volume is dimensionful, so the units of the integral are fb $\times$ GeV$^{2(n-2)}$.

2. Without option (default), the result of the integration is not a cross section, but the phase space volume, normalized by the massless phase space volume (4). Thus, if all final-state particles are massless and no cuts are specified, the *integral* is unity regardless of the number of particles.

Cuts and nonvanishing masses will affect the result accordingly. If events are generated, the final-state particles are distributed uniformly in phase space.

Note that the phase-space integration for the `test` matrix element is organized in the same way as it would be for the real $2 \to n$ process. Since such a phase space parameterization is not optimized for the constant matrix element that is supplied instead, good convergence is not guaranteed. (Setting `stratified = F` may be helpful here.)

The possibility to call a dummy matrix element in this way allows to histogram spectra or structure functions: Choose a trivial process such as $uu \to dd$, select the `test` method, switch on structure functions for one (or both) beams, and generate events. The distribution of the final-state mass squared reflects the x dependence of the selected structure function.

For the `test` method there are two more options:

`p`: : The letter `p` indicates that beam polarization should be transferred to the final state unchanged. (Otherwise, it would be ignored.) This is only possible if initial and final state coincide, e.g, $e^+e^- \to e^+e^-$. The option is useful for testing polarized spectra and structure functions.

```
ee_test_pol    e1,E1  e1,E1   test   p
```

`u`: : The letter `u` indicates that the matrix element is not just constant, but unity. Such code may serve as the initial setup for inserting some nontrivial matrix element by hand.

For historical and completeness reasons we mention here the technical details and available options when using `CompHEP` or `MadGraph` matrix elements.

The `CompHEP` version included in `WHIZARD` can only generate matrix elements for up to four particles in the final state. No polarization or flavor summation is possible (in contrast to the official `CompHEP` version), and color-flow information is not explicitly generated in the version used by `WHIZARD`. However, arbitrary models can be defined and the predefined models can be modified to include anomalous couplings etc. `CompHEP` matrix elements, evaluated by the trace method, are most efficient for simple $2 \to 2$ or $2 \to 3$ processes, while for $2 \to 4$ processes the other two packages typically generate faster code.

The `MadGraph` version inside `WHIZARD` has the Standard Model implemented with the exception of the trilinear or quartic Higgs couplings. The user may choose the maximal order of the strong (QCD), electromagnetic (QED), or weak (QFD) coupling for the matrix element evaluation. The implementation included in `WHIZARD` can generate matrix elements with up to six final-state particles. No flavor summation is possible.

There are several options in the fifth entry of the input file for the two generators: The letter `s` indicates that only a selection of particular Feynman diagrams is taken. Example:

```
nnh        e1,E1   n1,N1,H   chep
nnh_zh     e1,E1   n1,N1,H   chep   s
```

The setup of the necessary diagram selection file is described in Sec. 5.1.

For MadGraphonly it is possible to define the maximal order in each of the coupling constants: 'QCD' counts powers of g_s (gluon couplings), 'QED' counts powers of e (photon couplings), and 'QFD' counts powers of the remaining electroweak couplings. As an example, the following specifications generate the QCD and the electroweak contribution to $u\bar{u} \to d\bar{d}$ separately:

```
uudd_qcd   u,U   d,D    mad    n:QED<=0, n:QFD<=0
uudd_ew    u,U   d,D    mad    n:QCD<=0
```

There is only an upper bound on the number of couplings of a given species supported, so `<=` is the only available relation.

MadGraph always uses a fixed-width prescription, both in the s- and the t-channel, so there is no option. The "fudged" width option mentioned above can be activated for CompHEP by setting the flag `gwidth`. However, this is a runtime flag that enters the input file `whizard.in` in the block `parameter_input`.

4.2 Compilation, installation, and bundles

After the processes have been selected, the matrix elements and executable code are compiled and linked by the simple command

```
make prg
```

This will generate an executable named `whizard` in the `bin` subdirectory.

An alternative is

```
make bundle-src
make bundle
```

which will create a *restricted bundle* containing the Fortran 95 source code of the selected matrix elements and the libraries which have not been explicitly disabled.[22] This bundle will be put as a gzipped tar file in the main directory under the name `whizard-bundle-yymmdd-hhmm.tgz`, where `yymmdd-hhmm` specifies the current date and time. The bundle can be unpacked on a different platform. There, it can be configured, compiled and run without the need for a working `O'Mega` (i.e. `O'Caml`) installation. **Important:** In the bundle, you have to use the commands `make prg_bundle install_bundle` instead of `make prg install`, otherwise it won't work. (The necessary `O'Mega` library will be contained in the bundle, of course.) The configuration and input files (`whizard.prc`, `whizard.in`, etc.) will be copied from the `conf` directory of the distribution (not from the `results` directory). The site-specific configuration `config.site` will *not* be copied (precisely because it is site-specific). The restricted bundle has all capabilities of the `WHIZARD` system except for the possibility to re-generate the matrix elements.

If only the source code for the process list should be generated, say

```
make proc-src
```

[22] Those that are part of the `WHIZARD` system. Precompiled libraries such as the LHAPDF have to be present on the target system and can be enabled there if they are needed.

The files can be found in the subdirectory `processes-src`. If desired, the source code can be modified before a further `make` call compiles and links them together.

A run of the Monte Carlo generator requires, apart from the executable, the following input and configuration files. In their initial form they are found in the `conf` subdirectory:

- `whizard.in` containing all input data and parameters.
- `whizard.mdl` containing vertex definitions used in phase space setup. If an appropriate phase space configuration file `whizard.phs` exists, the model file is not necessary.
- `whizard.cut1` containing a-priori cut definitions. This file may be empty.
- `whizard.cut5` containing cut and histogram definitions for the data analysis step (the 5th pass of the program). This file may be empty.

When the command

```
make install
```

is issued, all necessary configuration files, together with the `whizard` executable, are copied into the `results` subdirectory. If no user-defined files are found, default ones will be used. The contents of the `results` directory may be copied anywhere without affecting the functionality of the `whizard` executable within, leaving room for the simultaneous use of multiple `WHIZARD` copies with different process content. ('`make install`' implies '`make prg`', so the latter is in fact redundant.)

Alternatively, one can omit the `install` step and copy the `WHIZARD` executable to any place in the system, or just leave it in the `bin` subdirectory. For a successful run of `WHIZARD`, one needs, of course, copies of the input files in the working directory.

Dealing with compilation problems

Although the authors have tried to make the program compile and run in various environments, it may happen that something goes wrong. The following problems are, at least, known:

- The `O'Caml` compiler is missing on the host computer. This is not a fatal problem since the compiler is part of most Linux distributions and available via MAC OS X `port`. Furthermore, it is easily available via http://pauillac.inria.fr/ocaml. Note that the `O'Caml` programming language can be installed on virtually any UNIX and MAC OS X system, and that it is perfectly possible to do this on a user account without superuser access.
- One of the PERL scripts which organize matrix element generation fails. This may be due to inconsistent shell return codes on some systems (IRIX, for instance). Since the OS world seems to converge towards Linux, this problem rarely occurs in reality. (Please notify the author if you really need a fix for a different OS. However, for `WHIZARD` 1 we basically consider Linux, MAC OS X and ALPHA as supported architectures.)

- A matrix element generator (`O'Mega`, `CompHEP`, `MadGraph`) fails. The reason is probably an attempt to generate a process that is not supported or which is physically impossible. [23] If this happens, there could be some trace in the temporary directory that has been created (in `/tmp` if TMPDIR has not been set). Note that you should remove this directory by hand if the program terminates abnormally during process creation.

- The Fortran 95 compiler fails. You may have found an actual `WHIZARD` bug, or you may have discovered a bug in the Fortran 95 compiler (this has happened more than once during the development of `WHIZARD`!). Make sure that you have the most recent compiler version and complain to the compiler vendor, if nothing else helps. Note however, that `WHIZARD` 1 explores no Fortran 03 features and most Fortran 95 are very mature nowadays).

- The linker fails due to incompatible object file formats. The reason could be that the `WHIZARD` configuration has chosen a Fortran 77 compiler different from the one used in generating the precompiled Fortran 77 libraries (`PYTHIA`, for instance). Set the correct compiler using the environment variable `F77`, reconfigure and recompile.

- The program crashes with a runtime error. This can be due to an incorrect input syntax, e.g., in a `NAMELIST` block. A frequent error is the specification of parameters that are misspelled or unknown to the current model. Another source of runtime errors can be incompatible external library formats (`PYTHIA` etc.). Otherwise, in particular if there is no reasonable error message, it may be a `WHIZARD` bug. Please report this to one of the email addresses below.

- Floating underflow messages after successful program execution are usually harmless.

- `CompHEP` cannot be compiled, so the executable `x_comphep` is missing when `WHIZARD` tries to generate matrix elements. This is likely due to missing C libraries needed by `CompHEP`. In particular, although `WHIZARD` will call `CompHEP` in command-line mode, the `CompHEP` executable needs the X libraries and include files installed on the host system. Note that `CompHEP` is just for self-testing purposes and is considered to be no longer maintained by us.

In any case, whether you could not solve a configuration or compilation problem or you found a workaround, please contact the author

`kilian@physik.uni-siegen.de`
`ohl@physik.uni-wuerzburg.de`
`juergen.reuter@desy.de`
`christian.speckner@physik.uni-freiburg.de`

so that the issue can be dealt with in a future release. (Please make sure that you are running the most recent version of `WHIZARD`, or at least look in the current CHANGES file whether a bug has been fixed in the meantime.)

[23] For `MadGraph` in particular, if there is an ambiguity you may have to set the QCD order as an option.

4.3 Process selection

For running WHIZARD, the user can freely choose among the processes he has defined in the configuration file and compiled in the executable. The parameter process_id in the main input file (see below) is a string which contains the list of process tags, e.g.

```
process_id = "zh, zz, zw"
```

If there is more than one process, integration of the cross sections will be performed in sequence. If a subsequent simulation step is requested, the events will be mixed according to their relative total cross sections. If fragmentation is enabled, one can even mix WHIZARD processes with processes generated by PYTHIA (see Sec. 4.8).

Since the integration grids are written to separate files, they can be reused later if events are to be simulated for the same or a different collection of processes, provided the input parameters and cuts are identical. This may save a lot of adaptation and integration time.

If the parameter process_id is left empty, WHIZARD will just print the list of processes, write a logfile containing the complete parameter list (including user settings), and exit.

4.4 Input data

When the whizard executable is started, it will use the parameters from the copy of the file whizard.in in the working directory. In this file, the physical input parameters and runtime switches are set. There are very many parameters, and all of them are optional. WHIZARD will try to insert sensible default values if the user has not specified a value. Of course, if no process tag is given, WHIZARD can do nothing, so it will just display the list of processes and exit.

The file whizard.in is written using the Fortran 95 NAMELIST conventions. It consists of several blocks marked by a keyword beginning with the & character and closed by a slash /. Each block contains a list of assignments of the form *variable = value*. Variables left out are assigned their default value. The assignments are separated by commas or newlines. Whitespace can be inserted anywhere. Comments can be inserted in this file; they begin with ! and extend up to the end of the line.

An example for the input file is shown in Fig. 4. In many cases, it suffices to fill the process_input block and leave all others empty. If polarization, beamstrahlung etc. are intended, the beam_input blocks must also be considered.

As an alternative to the NAMELIST format, WHIZARD supports the SUSY Les Houches Accord format for its input file. See Sec. 4.4.7 for details.

4.4.1 Files

The process_input block allows to specify a generic filename. This can be used to organize WHIZARD run data by giving specific filenames. For instance, to indicate the process energy in the filename for all output files, one may write

```
&process_input
 process_id = "nnh"
```

```
&process_input
process_id = "nnh"
sqrts = 500
luminosity = 100
/

&integration_input
calls =
  1  10000
  5  10000
  2  20000
/

&simulation_input
write_events = T
/

&diagnostics_input /

&parameter_input
Me = 0
Ms = 0
Mc = 0
MH = 115
wH = 0.3228E-02
/

&beam_input
/

&beam_input
/
```

Figure 4: *Sample input file.*

```
 sqrts = 800
 filename = "whizard_800"
/
```

Then, the output files will be `whizard_800.out` and `whizard.800.nnh.out`, etc. You do not need to write a separate file `whizard_800.in`, however: if a specific file is not found, the program will always look for the default one, in this case `whizard.in`.

Similarly, the working directory can be specified by the `directory` parameter. Of course, the first input file to be read must reside in the initial directory, otherwise it cannot be found (unless you define it on the command line, see Sec. 5.6).

The `filename` setting is overridden by specific filenames such as `input_file`, or e.g. in the block `integration_input` the variables `read_phase_space_file` or `write_phase_space_file` (cf. below). Such filenames will be interpreted relative to the chosen `directory` (if set), unless

they begin with a slash, e.g. for `read_model_file` within the `integration_input` block [24],

```
read_model_file = "/home/whizard/standard_model/whizard"
```

or with "`./`", e.g.,

```
read_model_file = "./whizard"
```

The latter form refers to the *initial* working directory. Note that in all cases the filename extension (here, `.mdl`) will be appended to the filename you specify.

It is possible to specify an additional `input_file` name in the `process_input` block. This input file will be read after the current one. New settings of parameters override old ones. In principle, any number of input files can be read consecutively and merged in this way. Reading input is finished when an input file does not exist, or neither `directory` nor `input_file` is changed in the last reading. Obviously, one should avoid loops in file reading.

All external file names used by `WHIZARD` can be similarly redefined. As a special rule, the filename for reading something (cuts, grids, events etc.) will automatically be set equal to the corresponding filename for writing, unless it is specified explicitly. The reason is that those files will be rewritten if the corresponding input file is not found, which could lead to accidentally overwriting files which is not intended.

4.4.2 The `process_input` block

These parameters should always be inspected.

Parameter	Value	Default	Description
`process_id`	*string*	*empty*	Process tag(s) as defined in `whizard.prc`. It should contain the list of processes to activate, separated by commas or blanks, enclosed in quotes.
`cm_frame`	T/F	T	If true, the c.m. frame is the lab frame, the beams are in $\pm$ z directions, and the total c.m. energy is given by `sqrts`. If false, the beam energies and directions must be specified below in the blocks `beam_input`.

[24] Note that if `WHIZARD` does not find the corresponding file it takes its default file.

Parameter	Value	Default	Description
`sqrts`	*number*	0	If this number is greater than the sum of the incoming particle masses, it specifies the c.m. energy of the initial state in GeV. Applies only if `cm_frame` is true, and is ignored for decay processes.
`luminosity`	*number*	0	Integrated luminosity in fb^{-1}. A nonzero value will activate event generation. (Alternatively, the number of events can be specified below in the block `simulation_input`.) The `luminosity` value is ignored for decay processes.
`polarized_beams`	T/F	F	If true, the helicity content of the beams must be specified below in the blocks `beam_input`.
`structured_beams`	T/F	F	If true, the nature of the incoming beams must be specified below in the blocks `beam_input`.
`beam_recoil`	T/F	F	If true, and if structure functions (e.g., ISR) are selected, the recoil of the partons against the beam remnant (e.g., the emitted photons) is taken into account. The p_T distribution is computed within the approximation valid for the emission, hence it will be accurate at low p_T.
`recoil_conserve_momentum`	T/F	F	Applies only if `beam_recoil` is set: if true, keep momentum balance between parton and recoil momenta at the expense of energy balance. If false, keep energy balance at the expense of momentum balance.

Parameter	Value	Default	Description
`filename`	*string*	*empty*	Base filename (w/o extension) to be used for all input/output files instead of the string `"whizard"`.
`directory`	*string*	*empty*	Working directory for all further reading/writing of files.
`input_file`	*string*	*empty*	If nonempty, read the specified input file after the current one. The extension `.in` will be appended to the filename.
`input_slha_format`	T/F	F	If true, assume that the next input file is in SUSY Les Houches Accord format (see Sec. 4.4.7). If false, determine the format from the first line.

4.4.3 The `integration_input` block

These parameters will not affect the cross section value (with the possible exception of the default cut parameters) but can improve or disprove the running time, the stability and accuracy of the result, and the efficiency of event generation.

Parameter	Value	Default	Description
`calls`	*6 numbers*	*(yes)*	Array describing the number of iterations and number of calls per integration pass. Default values depend on the selected process. See below in subsection 4.6 for details.
`seed`	*integer*	*undefined*	Random number generator seed (integer). When omitted, the time counter will be used, resulting in a different value each run.
`reset_seed_each_process`	T/F	F	Reset the random number generator seed to `seed` not just once, but each time a process is integrated. This is useful for comparing matrix elements which should be identical, but, e.g., have been generated by different programs.

Parameter	Value	Default	Description
`accuracy_goal`	*number*	0	Goal for the accuracy estimate (6th column in the output). When this goal is reached and the efficiency goal is either also reached or unset, further grid adaptation iterations will be skipped (Sec. 4.9).
`efficiency_goal`	*percentage*	100	Goal for the reweighting efficiency estimate (7th column in the output). When this goal is reached and the accuracy goal is either also reached or unset, further grid adaptation iterations will be skipped (Sec. 4.9).
`time_limit_adaptation`	*integer*	0	If nonzero, grid adaptation for the current process will be stopped after the specified number of minutes, and the final integration pass started (Sec. 4.9).
`stratified`	T/F	T	Use stratified (T) / importance (F) sampling.
`use_efficiency`	T/F	F	Use efficiency (T) / accuracy (F) as the criterion for adapting the channel weights.
`weights_power`	*number*	0.25	Power used for adapting the channel weights. Lower value means slower adaptation (to suppress fluctuations).
`min_bins`	*integer*	3	Minimal number of bins per integration dimension.
`max_bins`	*integer*	20	Maximal number of bins per integration dimension. This number will be used as long as there are enough sampling points, otherwise the number of bins will be decreased.
`min_calls_per_bin`	*integer*	10	Minimal number of points per bin, integration dimension, and integration channel. If this limit cannot be satisfied, the total number of points will be increased.

Parameter	Value	Default	Description
`min_calls_per_channel`	*integer*	0	All integration channel will get at least (approximately) this number of points. Prevents channels from being dropped during adaptation.
`write_grids`	T/F	T	Write grids to files `whizard.grb` (best grid) and `whizard.grc` (current grid), to be reused later.
`write_grids_raw`	T/F	F	Use binary format for writing grids. Saves memory at the expense of portability.
`write_grids_file`	*string*	*empty*	If nonempty, use the specified filename for writing grids instead of the default. The file extensions are appended to *string*.
`write_all_grids`	T/F	F	Write, in addition, after each iteration the current grid to file `whizard.grXXX`, where `XXX` is the iteration number.
`write_all_grids_file`	*string*	*empty*	If nonempty, use the specified filename for writing the extra grids instead of the default. The file extensions are appended to *string*.
`read_grids`	T/F	F	Read existing grids `whizard.grb` and `whizard.grc` if they have been written by a previous run. This avoids the time-consuming adaptation step. Makes sense only if no physical parameters have been changed.
`read_grids_raw`	T/F	F	If true, search first for binary grid files, then for ASCII grids. If false, do search first for ASCII..
`read_grids_force`	T/F	F	Set this to T if you want to read the grids from file even if some parameters have changed. Use with care! This may result in a program crash if the grid structures are incompatible.

Parameter	Value	Default	Description
`read_grids_file`	*string*	*empty*	If nonempty, use the specified filename for reading grids instead of the default. The file extensions are appended to *string*.
`generate_phase_space`	T/F	T	Generate a phase space configuration appropriate for the current process and write it to `whizard.phx`.
`read_model_file`	*string*	*empty*	If nonempty, read vertex definitions for phase space setup from *string*`.mdl` instead of the default `whizard.mdl`.
`write_phase_space_file`	*string*	*empty*	Write phase space configuration to *string*`.phx` instead of the default.
`read_phase_space`	T/F	T	Read phase space configuration from `whizard.phs` or a previously generated file `whizard.phx` if possible.
`read_phase_space_file`	*string*	*empty*	Read phase space configuration from *string*`.phs` or *string*`.phx` instead of the default.
`phase_space_only`	T/F	F	Stop the program after phase space generation.
`use_equivalences`	T/F	T	If true, use permutation symmetry when updating grids to improve the quality of the results.
`azimuthal_dependence`	T/F	F	If false, it is assumed that the scattering does not depend on the overall azimuthal angle. This will be automatically T if general beam polarization is switched on, therefore the user need only access this parameter in the case of azimuthal-dependent cuts.

Parameter	Value	Default	Description
`write_phase_space_channels_file`	*string*	*empty*	Show phase space channels in *string*`.ps` instead of the default file `whizard-channels.ps`. Note that you need to call `CHANNELS=` *string* `make -e channels` in order to generate *string*`.ps`

The following parameters affect the details of the algorithm by which `WHIZARD` generates the phase space setup. Under normal circumstances, there should be no need to change them.

Parameter	Value	Default	Description
`off_shell_lines`	*integer*	1	Maximum number of off-shell-lines allowed for Feynman graphs which are initially taken into account for the phase space configuration. Log-enhanced (massless) propagators are not counted as off-shell.
`extra_off_shell_lines`	*integer*	1	Use configurations with more off-shell lines, if they happen to be maximally resonant.
`splitting_depth`	*integer*	1	Up to this number of branchings, a (massless) propagator will be considered as log-enhanced and mapped like a photon propagator.
`exchange_lines`	*integer*	3	Up to this number of t-channel propagators, a multiperipheral graph will be taken into account.
`show_deleted_channels`	T/F	F	With `extra_off_shell_lines`, extra channels will be generated which are deleted if they do not contain enough resonances. With this flag, they are just commented out, so they could be manually activated.
`single_off_shell_decays`	T/F	T	Whether single-off-shell decays are relevant for the phase space configuration.
`double_off_shell_decays`	T/F	F	Whether double-off-shell decays are relevant for the phase space configuration.

Parameter	Value	Default	Description
`single_off_shell_branchings`	T/F	T	Whether single-off-shell branchings are relevant for the phase space configuration.
`double_off_shell_branchings`	T/F	T	Whether double-off-shell branchings are relevant for the phase space configuration.
`massive_fsr`	T/F	T	Whether the radiation of a massive particle in the final state is relevant for the phase space configuration.
`threshold_mass`	*number*	50	A particle with a mass up to this value will be considered as massless for the purpose of phase-space setup. (But the true mass is taken into account when the particle appears as a resonant intermediate state.)
`threshold_mass_t`	*number*	200	A particle with a mass up to this value will be considered as massless for the purpose of phase-space setup, when it appears as a t-channel propagator.
`initial_decays_fatal`	T/F	T	As the phase space maps cannot describe on-shell decays of beam particles properly, WHIZARD normally gives a fatal error when such configurations are encountered. This option changes this to a warning at the price of a potentially screwed phase space setup.

The following parameters affect the setup of kinematical cuts. Note that the default cuts are taken into account only if there is no appropriate entry in the user cut file `whizard.cut1`. However, even if such an entry exists, the cut parameters below affect the phase-space mappings, so adjusting them to order-of-magnitude may improve the convergence of the result.

Parameter	Value	Default	Description
`default_jet_cut`	*number*	10	The default invariant mass cut in GeV applied to pairs of massless colored particles.
`default_mass_cut`	*number*	10	The default invariant mass cut in GeV applied to pair production of massless colorless charged particles and to photon emission.

Parameter	Value	Default	Description
`default_energy_cut`	*number*	10	The default energy cut in GeV applied to photon and gluon emission.
`default_q_cut`	*number*	10	The default Q cut in GeV applied to photon and gluon exchange.
`write_default_cuts_file`	*string*	*file*	If nonempty, write the list of default cuts to this file (augmented by the file extension) instead of the default. Note that the settings in this file are overwritten by a user-defined cut configuration, if present.
`read_cuts_file`	*string*	*empty*	Look for user-defined cut configurations in *string*`.cut1` instead of `whizard.cut1`.
`user_cut_mode`	*integer*	0	Set this nonzero to activate a user-defined cut function (Sec. 5.4).
`user_weight_mode`	*integer*	0	Set this nonzero to activate a user-defined weight function (Sec. 5.5).

4.4.4 The `simulation_input` block

These parameters control event generation, hadronization and output. If the luminosity (above) is left undefined or zero, one may still generate a fixed number of events by setting `n_events` nonzero. By default, the events are written to file in raw format and can be reused in that way. Writing events in another format (which uses up more disk space) must be requested explicitly. However, if this has not been done, one may use `WHIZARD` afterwards as a translator to any file format by reading pre-generated events and immediately writing them in another format:

```
&integration_input  read_grids = T /
&simulation_input   read_events = T  write_events = T /
```

`WHIZARD` also contains an experimental version of a k_T-ordered parton shower, written by Sebastian Schmidt. In the version it is implemented here, it is highly experimental and does work only for LHEF event formats (cf. below, this is the only one which allows for a varying number of particles in the event). A much more advanced version can be found in `WHIZARD` 2. For more details about the shower confer the `WHIZARD` 2 manual. The shower can be switched on with the `shower` flag.

Parameter	Value	Default	Description
`n_events`	*integer*	0	Number of (unweighted) events to generate at least, irrespective of the luminosity setting.

Parameter	Value	Default	Description
`n_calls`	*integer*	0	Number of matrix-element calls (weighted events) to execute at least, irrespective of the luminosity setting.
`n_events_warmup`	*integer*	0	Number of extra warmup events (see below, Sec. 4.7).
`unweighted`	T/F	T	Reweight events to generate an unweighted event sample.
`normalize_weight`	T/F	T	If true, normalize the event weight to unity. If false, normalize to the total cross section.
`write_weights`	T/F	F	If `unweighted=F`, write weight distribution data to `whizard.wgt`.
`write_weights_file`	*string*	*empty*	Write weight distribution to *string*`.wgt` instead.
`safety_factor`	*number*	1	Multiply the estimate for the highest weight by this factor before starting event generation.
`write_events`	T/F	F	Write generated events to file `whizard.evt` to be used by an external analysis package.
`write_events_format`	*integer*	1	The format to be used for writing events, where the file extension depends on the format (`.evt` for format = 1, see Sec. 4.7).
`write_events_file`	*string*	*empty*	If nonempty, use *string* as filename for writing events, where the file extension will be appended.
`events_per_file`	*integer*	0	If positive, begin a new event file once the number of entries exceeds this number. The event file counter is appended to each event file name, separated with a dot (before the file extension). This feature applies only to non-binary event formats.

Parameter	Value	Default	Description
bytes_per_file	*integer*	0	If positive, begin a new event file once the number of bytes in the file exceeds this number. The event file counter is appended to each event file name, separated with a dot (before the file extension). This feature applies only to non-binary event formats. See Sec. 4.7.1.
min_file_count	*integer*	1	If event files are split, use this index for the first event file. Increase the counter by one for each successive event file.
max_file_count	*integer*	999	Limit for the event file counter; if this limit is exceeded, event generation is terminated. (For weighted events only, this is an error condition since the event sample must be complete for being usable.)
write_events_raw	T/F	T	Write events to whizard.evx in condensed binary format, so they can be internally reused in another run.
write_events_raw_file	*string*	*empty*	Write raw events to *string*.evx instead.
read_events	T/F	F	Read events from file whizard.evx (raw format) instead of generating them. This is equivalent to read_events_raw.
read_events_force	T/F	T	This was intended to force WHIZARD to read in events from file even if some parameters have changed. However, the MD5 checksum implemented to check for parameter changes has some deficiencies. Hence, we always enforce to read in events, because this feature could otherwise not be used at all. Use with great care! (Note that this problem has been solved in WHIZARD 2.)

Parameter	Value	Default	Description
`read_events_raw_file`	*string*	*empty*	Read raw events from *string*`.evx` instead.
`keep_beam_remnants`	T/F	F	Keep the beam remnants in the event record when applying structure functions. See Sec. 4.4.8.
`keep_initials`	T/F	F	Keep the beam particles and the partons which initiate the hard scattering in the event record. See Sec. 4.4.8.
`guess_color_flow`	T/F	F	Infer the color flow for hadronization from the particle ordering, if it is nontrivial and not available directly.
`recalculate`	T/F	F	Recalculate the matrix element value for each event of a previously generated sample. Setting this flag automatically turns on reading grids and events from file.
`fragment`	T/F	F	Fragment the events depending on the value of `fragmentation_method` (see Sec. 4.8).
`fragmentation_method`	*integer*	1	Method used for fragmentation if `fragment` is true: 0=no fragmentation; 1=`JETSET`; 2=`PYTHIA`; 3=user.
`user_fragmentation_mode`	*integer*	0	When user-defined fragmentation routines are called, this parameter may select different modes.
`pythia_parameters`	*string*	*empty*	String to be given to `PYTHIA`'s `pygive` call before starting event generation. This allows to modify `PYTHIA`/`JETSET` properties, set particle masses, etc. The string is also available within user-defined fragmentation routines and can there be abused for different purposes.

Parameter	Value	Default	Description
`pythia_processes`	*string*	*empty*	`PYTHIA` background processes to be simulated in addition to the `WHIZARD` processes: A list of integers separated by blanks, enclosed in quotation marks. Refer to the `PYTHIA` manual for the list of processes.
`shower`	T/F	F	Switches the internal shower on or off. As a default it is off. Note that the shower only works with LHEF event format (foramt type 6). This shower is highly experimental. Mainly intended for testing purposes.
`shower_nf`	*integer*	5	Number of light flavors in the shower.
`shower_running_alpha_s`	T/F	F	Whether to use a running strong coupling α_s or not in the shower. As a default it is fixed.
`shower_alpha_s`	*number*	0.2	The value of the strong coupling constant α_s as used by the internal shower. The default is 0.2.
`shower_lambda`	*number*	0.29	The value of the QCD scale Λ_{QCD} as used by the internal shower. The default is 0.29 GeV.
`shower_t_min`	*number*	1.0	The infrared cutoff for the evolution parameter t_{min} as used by the internal shower. The default is 1 GeV.
`shower_md`	*number*	0.330	The constituent d quark mass m_d as used by the internal shower. The default is 0.330 GeV.
`shower_mu`	*number*	0.330	The constituent u quark mass m_u as used by the internal shower. The default is 0.330 GeV.
`shower_ms`	*number*	0.500	The constituent s quark mass m_s as used by the internal shower. The default is 0.5 GeV.
`shower_mc`	*number*	1.5	The constituent c quark mass m_c as used by the internal shower. The default is 1.5 GeV.

Parameter	Value	Default	Description
`shower_mb`	*number*	4.8	The constituent b quark mass m_b as used by the internal shower. The default is 4.8 GeV.

4.4.5 The `diagnostics_input` block

These parameters do not affect the result, but the information displayed on screen and stored in files.

Parameter	Value	Default	Description
`chattiness`	*integer*	4	How much information to show on screen: (0) only fatal errors, (1) and non-fatal errors, (2) and warnings, (3) and messages, (4) and results, (5) and debugging messages (if any).
`catch_signals`	T/F	T	If the compiler supports it, try to catch external signals such as SIGINT and SIGXCPU and exit gracefully, closing files first.
`time_limit`	*integer*	0	If nonzero, exit gracefully after the given number of minutes has passed. This is useful to prevent an external kill within a batch environment.
`warn_empty_channel`	T/F	F	Issue a warning whenever the integral within a phase space channel is zero.
`screen_events`	T/F	F	Whether to show generated events on screen.
`screen_histograms`	T/F	F	Whether to show histograms on screen.
`screen_diagnostics`	T/F	F	Whether to repeat the input parameters on screen.
`show_pythia_banner`	T/F	T	Whether to display the `PYTHIA` banner page if fragmentation is enabled.
`show_pythia_initialization`	T/F	T	Whether to display the `PYTHIA` initialization messages if fragmentation is enabled.
`show_pythia_statistics`	T/F	T	Whether to display the `PYTHIA` statistics summary after event generation is completed.

Parameter	Value	Default	Description
`write_logfile`	T/F	T	Whether to write the (process-specific) output file(s) `whizard.XXX.out`.
`write_logfile_file`	*string*	*empty*	Use this as the filename for the logfile.
`show_input`	T/F	T	Whether to repeat the input parameters in the logfile.
`show_results`	T/F	T	Whether to show the integration results in namelist format in the logfile.
`show_phase_space`	T/F	F	Whether to show the phase space configuration in the logfile.
`show_cuts`	T/F	T	Whether to show the cut configuration in the logfile.
`show_histories`	T/F	F	Whether to show the individual `VAMP` channel histories in the logfile.
`show_history`	T/F	T	Whether to show the overall `VAMP` history in the logfile.
`show_weights`	T/F	T	Whether to show the weight adaptation in the logfile.
`show_event`	T/F	F	Whether to show the last event in the logfile.
`show_histograms`	T/F	F	Whether to show histograms in the logfile.
`show_overflow`	T/F	F	Whether to show events beyond the first or last bin in histogram listings.
`show_excess`	T/F	T	Whether to show a summary of events with weight exceeding one.
`read_analysis_file`	*string*	*empty*	Use this (*string.cut5*) as the filename for the analysis setup instead of `whizard.cut5`
`plot_width`	*number*	130	The width in mm of the plots if online analysis is enabled.
`plot_height`	*number*	90	The height in mm of the plots if online analysis is enabled.
`plot_excess`	T/F	T	In the plots, display excess events in red.
`plot_history`	T/F	T	If this is enabled, write a graphics driver file for displaying the integration history, i.e., the integral with error bars for each iteration. Use `make history` to generate the graphics file `whizard-history.ps`.

Parameter	Value	Default	Description
`plot_grids_channels`	*string*	*empty*	The string is a list of phase-space channels (integers) for which the bin distribution will be histogrammed. Use `make grids` to generate the graphics file `whizard-grids.ps`.
`plot_grids_logscale`	*number*	10	Use logarithmic scale for the grid plots if the bin width varies over more than this ratio.
`slha_rewrite_input`	T/F	T	If SUSY Les Houches Accord data have been used, whether to repeat this input in the process-specific logfiles (including comments) or to rewrite it there using the data which have actually been used.
`slha_ignore_errors`	T/F	F	If this is false, an error signaled in the SLHA input file (in the SPINFO or DCINFO block) will cause `WHIZARD` to stop before calculating anything. If true, such errors will be displayed, but the run continues.

4.4.6 The `parameter_input` block

These are the physical constants used in evaluating the matrix elements. If this block is left empty, default values (see below) will be inserted.

Which constants are actually present, and which ones are dependent or derived, depends on the physical model. Consult the model files in `conf/models` for the corresponding parameters. For example, the constants relevant for the Standard Model `SM_CKM` are shown below together with their default values:

Parameter	Value	Default	Description
`GF`	*number*	1.16639×10^{-5}	Fermi constant
`mZ`	*number*	91.1882	Z-boson mass
`mW`	*number*	80.419	W-boson mass
`mH`	*number*	200	Higgs mass
`alphas`	*number*	0.1178	Strong coupling constant $\alpha_s(M_Z)$
`me`	*number*	0.000511	electron mass
`mmu`	*number*	0.1057	muon mass
`mtau`	*number*	1.777	τ-lepton mass
`ms`	*number*	0.12	s-quark mass
`mc`	*number*	1.25	c-quark mass

Parameter	Value	Default	Description
`mb`	*number*	4.2	b-quark mass
`mtop`	*number*	174	t-quark mass
`wtop`	*number*	1.523	t-quark width
`wZ`	*number*	2.443	Z-boson width
`wW`	*number*	2.049	W-boson width
`wH`	*number*	1.419	Higgs width
`vckm11`	*number*	0.97383	Vud
`vckm12`	*number*	0.2272	Vus
`vckm13`	*number*	0.00396	Vub
`vckm21`	*number*	-0.2271	Vcd
`vckm22`	*number*	0.97296	Vcs
`vckm23`	*number*	0.04221	Vcb
`vckm31`	*number*	0.00814	Vtd
`vckm32`	*number*	-0.04161	Vts
`vckm33`	*number*	0.99910	Vtb
`khgaz`	*number*	1.000	anomaly Higgs coupling K factors
`khgaga`	*number*	1.000	anomaly Higgs coupling K factors
`khgg`	*number*	1.000	anomaly Higgs coupling K factors

The dependent parameters (such as e, $\sin\theta_W$, V_{cb} etc.) will be shown in the output file `whizard.out`, but one should not try to reset them in the input file. The dependencies take into account the tree approximation only, such that gauge invariance is automatically respected. This may lead to unwanted side-effects in particular cases, e.g., the Higgs coupling to b quarks is proportional to the b mass, so setting $m_b = 0$ removes all $H \to b\bar{b}$ subprocesses.

Setting fermion masses to zero will considerably speed up the matrix element evaluation since certain helicity combinations vanish identically (with `O'Mega` and `MadGraph`). In that case, cuts may be needed for a finite answer. However, if necessary, finite masses can be kept for all particles except the light quarks u and d.

Concerning quark masses in particular, they depend in fact on the chosen scale. The default values for the parameters include the strong coupling constant defined at the scale $\mu = M_Z$, but the quark masses evaluated at the low scale $\mu = 2$ GeV. These default values are taken from the PDG 2000 compilation [25]. For a consistent scheme, running quark masses should be inserted instead wherever this is of importance. For instance, the b mass is only about 2.9 GeV at $\mu = M_Z$, thus affecting the Higgs coupling to b quarks.

Although they can be calculated from the other parameters in principle, the particle widths are retained as independent input parameters. Thus, the branching ratio of a particular resonance decay can be tuned by modifying the resonance width. When the Higgs mass is changed, the Higgs width has to be changed accordingly if Higgs decays are considered. The default values for the particle widths are given by the sum of the tree-level $1 \to 2$ decay channels, using the default masses and couplings as input. This choice is questionable as well. Since

[25] Note that we do not keep this updated. The parameter file of version 2, however, contain the most recent PDG values

O'Mega(MadGraph and CompHEP, too) can only calculate cross sections at tree level, this is at least consistent; however, for each particular problem one should re-investigate the underlying assumption that 2-particle decays are dominant, and recalculate the widths accordingly. This cannot be done automatically by WHIZARD 1.

Finally, even if a parameter is defined and appears in the output, this does not necessarily mean that its value is used by all three programs CompHEP, MadGraph and O'Mega. For instance, the CKM matrix is not taken into account by the MadGraph version of the Standard Model.

For using CompHEP, there are two switches within the parameter_input block:

Parameter	Value	Default	Description
gwidth	T/F	F	Use the gauge-invariant width prescription for unstable particles in CompHEP matrix elements. This is the so-called overall factor scheme. For O'Mega, the corresponding 'f' option has to be specified in the process configuration file already.
rwidth	T/F	F	Use a running-width prescription for unstable particles in CompHEP matrix elements.

4.4.7 Input in SUSY Les Houches Accord format

The SUSY Les Houches Accord (SLHA) [10,11] describes a format to pass MSSM/NMSSM physics data between computer programs. WHIZARD accepts its input file in SLHA format. This requires that either

- The first line of the input file begins with the string

  ```
  # SUSY Les Houches Accord
  ```

- or the parameter input_slha_format in the process_input block (see above) is .true. This flag can be set on the command line, or in an input file which is read before the SLHA data.

It is possible to

- read SLHA input after the usual input file: In the process_input block of the standard input file, set input_file to the name of the SLHA file (without the extension .in which each input file must have).

- embed standard WHIZARD input within the SLHA file: Enclose it in a block WHIZARD_INPUT. The text of this block is read by WHIZARD only and copied verbatim into an internal file which is read after the SLHA data. Note that, according to the SLHA standard, the first character of each line within this block has to be blank.

In this way, several input files in both formats may be read consecutively, which could be useful to set defaults and override some parameters computed by a SUSY spectrum calculator.

However, the recommended procedure is to supply a standard `whizard.in` file where the process selection, collider information, and `WHIZARD` switches and runtime parameters are set, while containing all physical data in a separate file written in SLHA format which is read afterwards. The latter may be produced by an automatic spectrum and decay calculator.

In the output, if SLHA data have been used, the data will be repeated. `WHIZARD` reorders the blocks, where the blocks that actually have been used (which may include `MODSEL`, `SMINPUTS`, `MINPAR`, `MASS`, and mixing data) come first. The parameter `slha_rewrite_input` in the `diagnostics_input` block controls whether the input data are just copied into the output (preserving all comments, but not necessarily the ordering of blocks) or rewritten by `WHIZARD`. In the latter case, the relevant parameters in the output are made consistent between the standard `WHIZARD` format and the SLHA format, where applicable. Furthermore, for each process a block `PROCESS` will be appended which contains the integration results in a format which follows the SLHA conventions. The block `CSINFO` (cross section calculator information) is added, analogous to the `SPINFO` and `DCINFO` blocks described in the standard (see Fig. 5).

`WHIZARD` will use the `MASS` and `DECAY` information in the SLHA file to set the physical masses and widths of the particles it recognizes. Branching ratio information is not used. Note that the masses of some SM particles (Z, t, b, and τ) are defined in SMINPUTS. For completeness, they may also be set in the `MASS` block (where the W mass is located). If the values disagree, the `MASS` information takes precedence for `WHIZARD`'s needs. The widths of Z, W, and t may be set using the `DECAY` format. In all cases, default values (which may have been set in a previous input file) will be taken without warning where data are missing.

While the SLHA is intended for the MSSM and its extensions, it is possible to use this format to pass physical parameters for a SM calculation, too. If the SM is chosen when configuring `WHIZARD`, only the blocks `MODSEL`, `SMINPUTS`, `MASS` and `DECAY` information will be used. By convention, we define the `model` parameter (in the `MODSEL` block) to be negative for a non-SUSY model.

4.4.8 The `beam_input` blocks

The input file is finished by two blocks which describe the properties of the first and second incoming particle beam, respectively. It depends on the master flags in the `process_input` block above whether particular information within these blocks is actually used:

- If `cm_frame` is false, the `beam_input` blocks specify information on the beam energies and directions. If `cm_frame` is true (default), this information is discarded, the beam energies are given by the `sqrts` value, and the beam directions are assumed along the positive and negative z axis, respectively.

- If `polarized_beams` is true, the `beam_input` blocks specify information on the beam polarization. If `polarized_beams` is false (default), this information is discarded and the beams are assumed to be unpolarized.

- If `structured_beams` is true, the `beam_input` blocks specify information on the beam particle types, beam energy spectra and/or structure functions. If `structured_beams` is

false (default), the beam particles are assumed to be the incoming particles of the hard process, and the partonic energies are set equal to the beam energies.

If all of the master flags are kept at their default values, the content of the `beam_input` blocks is completely discarded, so they may be left empty.

For decay processes, there are some obvious restrictions: There is only one "beam", so the second block is ignored. While `cm_frame` and `polarized_beams` retain their meaning (i.e., one can simulate the decay of a moving and/or polarized particle), `structured_beams` is ignored, and no structure functions can be selected.

Energy and direction If the master flag `cm_frame` is false, the beam energies and directions are set by

Parameter	Value	Default	Description
`energy`	*number*	0	If greater than the beam particle mass, this specifies the beam energy in the lab frame. Otherwise, the beam energy is set equal to the particle mass (fixed target).
`angle`	*number*	0	If `direction` is not set, this specifies a rotation of the beam axis in the lab frame around the positive y axis. (By default, the beam directions are along the positive/negative z axis, so a rotation by the angle $\pi/2$ turns them into the positive/negative x axis.) If `direction` is set, this parameter is ignored.
`direction`	*three numbers*	0 0 0	If any component is nonzero, this vector explicitly specifies the direction of the given beam in the lab frame.

Polarization If the master flag `polarized_beams` is true, the polarization is set for each beam by

Parameter	Value	Default	Description
`vector_polarization`	T/F	F	If false (default), use the standard helicity basis (left-/right-handed). Set this flag if you need another basis, in particular transversal polarization.

Parameter	Value	Default	Description
`polarization`	*two (three) numbers*	0 0 0	Fraction of left/right polarization (fermions, photons, gluons), resp. left/longitudinal/right polarization (massive vector bosons). If the vector polarization model is selected, the three numbers denote the polarization vector.

For fermions, the default (helicity basis) polarization model describes longitudinally polarized beams with the specified fraction of left- and right-handed particles. In the case of massless vector bosons, this corresponds to left and right circular polarization along the beam axis. Massive vector bosons have three entries, where the middle entry specifies the fraction of longitudinal polarization. If the numbers sum up to less than 1, the remainder is filled with unpolarized particles. For instance, .5 0 and .75 .25 both stand for 50 % left-handed polarized electrons.

The quantization axis of a polarized particle is the momentum vector in the lab frame. For particles at rest, the quantization axis is the positive z axis.

The vector polarization model currently applies to fermions only. The three numbers denote the components of a three-vector $\vec{P}$ with length less or equal to 1. The helicity density matrix for fermions is given by

$$\rho = \tfrac{1}{2}(1 + \vec{P} \cdot \vec{\sigma}), \tag{5}$$

where the polarization vector has components (P_x, P_y, P_z). For antifermions this is replaced by

$$\rho = \tfrac{1}{2}(1 + \vec{P} \cdot \vec{\sigma}^*), \tag{6}$$

so the y component changes sign.

While the z component of $\vec{P}$ describes longitudinal polarization, the x and y components correspond to transversal polarization. Longitudinally polarized states can therefore be specified in both polarization models, while transversal polarization requires the vector model.

Spectra and structure functions If the master flag `structured_beams` is true, `WHIZARD` will check if any of the structure functions defined below is applicable for this beam. For each beam, the beam particle is defined by either one of

Parameter	Value	Default	Description
`particle_code`	*integer*	0	PDG code of the incoming beam particle.
`particle_name`	*string*	`"UNKNOWN"`	Name of the incoming beam particle.

If both entries are specified, the given PDG code takes precedence.

The further parameters specify which structure functions are used for each beam and fix structure function parameters where necessary. Structure functions are applied consecutively (Tab. 1), choosing a sensible chain of beam particle splittings: For instance, if the beam particle

Structure function	in-particle	out-particle	user beam pol.
File events	*any*	$= in$	*keep*
User spectrum (Sec. 5.3)	*any*	*any*	*keep/ignore*
CIRCE (beamstrahlung)	$e^\pm$	$e^\pm$	*keep*
	$e^\pm$	γ	*ignore*
CIRCE2 (beamstr.,γ spectra)	$e^\pm/\gamma$	$e^\pm/\gamma$	*ignore*
ISR (photon radiation)	charged particle	unchanged	*keep*
EPA (effective photon approximation)	charged particle	γ	*ignore*
EWA (effective W approximation)	weak doublet	weak doublet	*ignore*
LHAPDF (parton distribution functions)	$p, \bar{p}$ or γ	q or g	*ignore*
PDF (parton distribution functions)	$p, \bar{p}$ or γ	q or g	*ignore*
User structure function (Sec. 5.3)	*any*	*any*	*keep/ignore*

Table 1: *Available structure functions. Each transforms an in-particle into a collinear out particle with a certain momentum fraction. They may be concatenated in the order given in the table. In the present implementation, the user-specified beam polarization is either kept unchanged for all events, or is overwritten.*

is an electron and the process is initiated by a quark, switching on both EPA and PDF will take the quark content of a photon which is emitted (in the Weizsäcker-Williams approximation) by the incoming electron. `WHIZARD` will not check whether the selected combination makes physically sense.

If it is desired to have the beam remnants in the event record, one should set the parameter `keep_beam_remnants` in the `simulation_input` block. The beam remnants will be photons (beamstrahlung, ISR), charged particles (EPA) or undefined hadron remnants (LHAPDF,PDF). A beam remnant is always massless, regardless which mass the particle has been assigned somewhere else, and it will have no transverse momentum. There will be exactly one beam remnant per structure function even though the number of photons radiated in beamstrahlung and ISR is not well-defined in reality. Beam remnants cannot be used for the built-in analysis since they are assigned no binary codes. Their main virtue is to make possible a reconstruction of individual splittings in the initial state in an external analysis.

Furthermore, one can set `keep_initials`. In that case, the beam particles and the partons which initiate the hard scattering are kept in the beam record. Together with the beam remnants, this allows for full mother-daughter relationships to be displayed. For fixed-energy beams, there are no beam remnants, and the initial partons are identical with the beam particles.

Events from file `WHIZARD` is able to read a sample of events from file and use it for integration and event generation. If both beams are read from file in this way and the filename is identical for both beams, the program expects a pair of real numbers in each line of the file. (Otherwise, it expects separate files with a single real number per line.) Each number corresponds to the energy carried by the beam particle. Empty lines and comment lines (marked by `#` or `!`) are skipped.

When the end of the beam event file is reached, a warning is issued and the file is reread from the beginning. Note that reusing the event sample in this way leads to systematic errors in the calculation. For instance, the fluctuations of the computed cross section between subsequent iterations can exceed the error estimate by orders of magnitude. In the final integration step, this could be visible as a χ^2 value which is anomalously large. To avoid this problem, it is recommended to use an event sample which is statistically distributed and contains a sufficient number of events so that no rewinding is necessary.

Parameter	Value	Default	Description
`FILE_events_on`	T/F	F	Whether to read the beam events from file.
`FILE_events_file`	*string*	*empty*	File name. The directory prefix (as specified by `directory` above, if any) will be prepended and the extension `.bev` will be appended to this string. If this file does not exist, the default file `whizard.bev` is searched instead.
`FILE_events_energies`	T/F	T	Whether the numbers in the file should be interpreted as absolute energies (T) or as energy fractions with respect to the nominal beam energy, i.e., $\sqrt{s}/2$ in the c.m. frame (F).

Parton distribution functions There are two possibilities two access parton distribution functions inside hadrons: either via the `LHAPDF` library or via the `PDFLIB` inside the CERNLIB. Meanwhile we consider `LHAPDF` to be the default option, while we mention the `PDFLIB` setup just for backwards compatiblity and completeness. Distribution functions exist for all quarks and for the gluon. For `LHAPDF`, the parameters `LHAPDF_on`, `PDF_scale` (if `PDF_running_scale` is false) and `LHAPDF_file` for the PDF set are mandatory. For an overview over available sets and parameters confer the `LHAPDF` manual.

Parameter	Value	Default	Description
`LHAPDF_on`	T/F	F	Whether to use PDFs from LHAPDF.
`LHAPDF_file`	*string*	*empty*	File name for PDF set, e.g. `cteq6ll.LHpdf`
`LHAPDF_set`	*integer*	0	Number of the structure function set within the given file. Usually, 0 is the central value, if values other than zero are available they correspond to error sets.

Parameter	Value	Default	Description
PDF_running_scale	T/F	F	Whether to use a running scale (the total partonic invariant mass) for the structure function evaluation.
PDF_scale	*number*	0	If PDF_running_scale is false, this parameter must be set to a fixed energy scale for the structure function evaluation.

For PDFLIB inside the CERNLIB setup, the parameters are listed below (for a more detailed description, refer to the PDFLIB manual). For the numerical parameters, a zero value will cause the PDFLIB default value to be taken, so only PDF_on and PDF_scale (if PDF_running_scale is false) are mandatory.

Parameter	Value	Default	Description
PDF_on	T/F	F	Whether to use PDFs.
PDF_ngroup	*integer*	0	Number of author group. If left 0, the PDFLIB default value (MRS) will be used.
PDF_nset	*integer*	0	Number of the structure function set by the given group. If left 0, the PDFLIB default value will be used.
PDF_nfl	*integer*	0	Number of active flavors.
PDF_lo	*integer*	0	Order of α_s calculation.
PDF_running_scale	T/F	F	Whether to use a running scale (the total partonic invariant mass) for the structure function evaluation.
PDF_scale	*number*	0	If PDF_running_scale is false, this parameter must be set to a fixed energy scale for the structure function evaluation.
PDF_QCDL4	*number*	0	QCD scale in GeV for four active flavors.

Beamstrahlung (CIRCE1) For beamstrahlung, the accelerator type should always be defined, and it is not recommended to use an energy different from the values for which valid CIRCE [6] parameterizations exist. There is the option to either use the generator mode of CIRCE, or to accept the distributions calculated by CIRCE as analytical spectra. For efficiency reasons, the generator mode is typically preferred. An example input file whizard.in.circe_isr for the default process ee_o can be found the conf directory.

Parameter	Value	Default	Description
CIRCE_on	T/F	F	Whether to apply beamstrahlung (electron, positron or photon beam).

Parameter	Value	Default	Description
CIRCE_generate	T/F	T	Whether to use the CIRCE event generator or the analytical beamstrahlung spectra.
CIRCE_map	T/F	T	Whether to apply a mapping to improve convergence (relevant for analytical spectra only).
CIRCE_sqrts	*number*	$\sqrt{s}$	The reference energy. Note that CIRCE spectra are defined only for the discrete energy values (cf. the CIRCE write-up for more info) .
CIRCE_ver	*integer*	0	The CIRCE version number required.
CIRCE_rev	*integer*	0	The CIRCE revision number required.
CIRCE_acc	*integer*	0	The accelerator type as needed by CIRCE.
CIRCE_chat	*integer*	0	The level of CIRCE diagnostics.

In general, more information can be found in the CIRCE manual on the HepForge page:

http://projects.hepforge.org/whizard/circe1.pdf

Photon spectra (CIRCE2) CIRCE2 is intended as a replacement for CIRCE1. While the previous implementation provided smooth spectra, the new version, which has to be used for photon collider spectra where no CIRCE1 version is available, provides histogrammed spectra which are read from a data file. These data files are partially contained in the WHIZARD distribution, in the subdirectory circe2-src/data. For more information on the CIRCE2 generator and possible updates on the spectra confer the manual/documented source code available from the HepForge page:

http://projects.hepforge.org/whizard/circe2.pdf

There is the option to either use the generator mode of CIRCE2, or to accept the distributions calculated by CIRCE2 as (histogrammed) spectra. For efficiency reasons, the generator mode is typically preferred.

The following parameters can be defined (identical for both beams):

Parameter	Value	Default	Description
CIRCE2_on	T/F	F	Whether to use CIRCE2 for simulating spectra.
CIRCE2_generate	T/F	T	Whether to use the CIRCE2 generator or the (histogrammed) spectra.
CIRCE2_verbose	T/F	T	Whether to print CIRCE2 messages on screen.

Parameter	Value	Default	Description
`CIRCE2_file`	*string*	*empty*	The data file to read by `CIRCE2`.
`CIRCE2_design`	*string*	"*"	The accelerator design for CIRCE2.
`CIRCE2_polarized`	T/F	T	Use a polarized (T) or polarization-averaged (F) spectrum in the datafile.
`CIRCE2_map`	-1/0/1	-1	Apply a power mapping for a singularity at $x = 0$ or $x = 1$, or no mapping (-1). Relevant only for histogrammed spectra, irrelevant for the `CIRCE2` generator mode.
`CIRCE2_power`	*number*	2	Parameter for the mapping, if any.

If `CIRCE2` is switched on, the `CIRCE2_file` parameter must always be given. The filename will be interpreted relative to the current working directory. The accelerator design should be specified if the data file contains more than one dataset. (The default matches the last dataset within the data file.)

It is important to note that polarization treatment is different from the standard `WHIZARD` conventions when `CIRCE2` is used. The `CIRCE2` data files introduce fixed polarization distributions of the beams which originate from realistic simulations. Therefore, the `polarized_beams` switch and the `polarization` settings should not be used together with `CIRCE2` under normal circumstances. Instead, the appropriate data file with polarization should be taken and `CIRCE2_polarized` switched on. This will generate the appropriate polarized spectra, properly normalized.

On the other hand, if a polarization-averaged dataset is specified, `CIRCE2_polarized` should also be switched off.

Despite the fact that the polarization distributions are a fixed property of any given dataset, one can reweight the polarization contributions by hand. To this effect one can set the parameters `polarized_beams` and `polarization`. For instance, by switching on only one helicity combination, the properties of the corresponding cross section component can be studied. This feature should be used with care, since the results may be unphysical.

In the spectra mode (`CIRCE2_generate`=false), if no mapping is specified by the user, the program inserts a mapping at $x = 0$ (power = 3) for photons, and a mapping at $x = 1$ (power = 12) for electrons. This assumes that all particles originate from an incident e^- beam, where for electrons there is a strong singularity (in fact, a smeared-out delta function) at $x = 1$. For a different setup, e.g., a positron beam, the mapping parameters have to be adjusted by hand. Note that without the $x = 1$ mapping, the delta-function singularity for electrons will almost certainly be missed by the adaptive integration, and the result may stabilize for a wrong cross section.

A sample input file, `whizard.in.photoncollider`, for the assumed hard process $\gamma\gamma \to e^-e^+$ can be found in the `conf` subdirectory.

ISR spectrum The ISR spectrum for charged particles accounts for the leading-logarithmic effect of multiple photon emission. In the leading-logarithmic approximation, universal terms

of order ϵ^n with $\epsilon = \frac{\alpha}{\pi} \ln \frac{s}{m^2}$ can be absorbed in a structure function. The leading singularity at $x = 1$ can be calculated exactly, while the other terms of the structure function are known only to finite order.

The implementation in WHIZARD takes into account the LLA structure function ([7]), including terms up to order ϵ^n, where $n = 0, 1, 2, 3$, depending on the value of `ISR_LLA_order`. The default value is $n = 3$. The scale which enters the ISR spectrum is given by `ISR_Q_max`; the default is to take the collider c.m. energy.

Since α_{QED} is usually calculated from G_F as far as the hard scattering is concerned, it makes sense to reset it to $\frac{1}{137}$ here. The incoming particle mass (e.g, m_e) must be defined here only if it has been set to zero in the parameter input block (e.g., to speed up the calculation).

Parameter	Value	Default	Description
`ISR_on`	T/F	F	Whether to apply ISR (electron or positron beam).
`ISR_alpha`	*number*	$e^2/4\pi$	The value of α_{QED} to be used for the spectrum.
`ISR_m_in`	*number*	m	The mass of the incoming particle.
`ISR_Q_max`	*number*	$\sqrt{s}$	The hard scale which cuts off photon radiation.
`ISR_LLA_order`	*0/1/2/3*	3	The order of the leading-logarithmic approximation.
`ISR_map`	T/F	T	Whether to use a mapping of the singularity at $x = 1$ when evaluating the structure function (recommended; note that switching this off might even lead to an uncaught arithmetic exception).

You can find an example, `whizard.in.circe_isr` for the sample process `ee_o` in the `conf` subdirectory.

EPA spectrum The EPA spectrum is the Weizsäcker-Williams approximation for a real photon radiated off the incoming beam. Here, Q_{max} and m_X must be supplied, the other parameters are optional. (If the produced system is massive, m_X should be set equal or less to the mass sum of all produced particles.)

Parameter	Value	Default	Description
`EPA_on`	T/F	F	Whether to use the EPA spectrum (photon beam).
`EPA_map`	T/F	T	Whether to apply a mapping to improve convergence.
`EPA_alpha`	*number*	$e^2/4\pi$	The value of α_{QED} to be used for the spectrum.

Parameter	Value	Default	Description
`EPA_m_in`	*number*	m	The mass of the incoming beam particle.
`EPA_mX`	*number*	0	The lower cutoff for the produced invariant mass.
`EPA_Q_max`	*number*	0	The upper cutoff on the virtuality of the photon ($Q_{\max} > 0$).
`EPA_x0`	*number*	0	The lower cutoff on the energy fraction of the incoming photon
`EPA_x1`	*number*	1	The upper cutoff on the energy fraction of the incoming photon

Again, an example, `whizard.in.epa` for the sample process $\gamma\gamma \to e^-e^+$ named `aaee` can be found in the `conf` subdirectory.

EWA spectrum The EWA spectrum is equivalent of the Weizsäcker-Williams approximation for massive W and Z bosons radiated off the incoming beam. For details about the EWA and its implementation into `WHIZARD` confer [13].

Parameter	Value	Default	Description
`EWA_on`	T/F	F	Whether to use the EWA spectrum (W/Z beam).
`EWA_map`	T/F	T	Whether to apply a mapping to improve convergence.
`EWA_pT_max`	*number*	0	The upper cutoff on the p_T of the W/Z ($p_{T,\max} > 0$).
`EWA_x0`	*number*	0	The lower cutoff on the energy fraction of the incoming W/Z
`EWA_x1`	*number*	1	The upper cutoff on the energy fraction of the incoming W/Z

Again, an example, `whizard.in.ewa` for the sample process $W^+W^- \to ZZ$ named `wwzz` in the leptonic setup of a 3 TeV CLIC collider can be found in the `conf` subdirectory.

4.5 Cuts

`WHIZARD` allows for cuts in some standard kinematical variables. Cuts may be necessary in case the matrix element is infinite without them (which happens if massless particles are either exchanged or produced). In this case, `WHIZARD` will apply default cuts on the invariant mass of colored or charged particle pairs, on the energy of emitted photons or gluons, and on the momentum transfer to exchanged photons or gluons. The values of those cuts are controlled by parameters in the input file (see above).

Alternatively, the user may specify his own set of cuts by supplying an entry in the file `whizard.cut1`. If such an entry exists, the program will ignore the default cuts, and the user

is responsible for setting up his cuts in such a way that the cross section is still finite. The format of such an entry for the process $e^-e^+ \to e^-e^+\gamma$ is shown in Fig. 6. The contents of `whizard.cut0`, where the default cuts are logged by `WHIZARD` (if any), may be used as a template.

The entry is begun by the keyword `process` followed by the process tag (`eeg` in this case). If the same set of cuts should be applied to several processes, one may specify more than one tag per entry, e.g.

```
process uudd uuss uucc
    cut ...
```

The cut configuration file may contain any number of process entries. Comments are marked by the letter `!` or `#`.

For a given process, each cut is indicated by the keyword `cut`, followed by a code letter, the keyword `of`, one or two binary codes, and cut window specifications. The code letters are listed in Table 2. The binary codes indicate the combination of particle momenta to which the cut should be applied. As the comment lines in Fig. 6 suggest, the outgoing particles are given codes $1, 2, 4, \ldots$. Two additional codes are assigned to the incoming particles, where the first incoming particle has the highest number. Momentum sums (particle combinations) are defined by adding binary codes, such that the code 5, for instance, stands for the sum of momenta of particles 1 and 4.

The initial momenta are always counted negative, such that (in Fig. 6) the number 17 stands for the *difference* of the momenta of particle 1 (outgoing) and 16 (incoming). The same applies to angles: The polar angle between 1 and 16 is not `A of 1 16` but `A of 1 8` since the initial particle 16 goes into the direction of $-p(8)$.

By default, the inserted momenta are those of the elementary partonic process. Thus, one can assume momentum conservation. For instance, in a $2 \to 2$ scattering process, the invariant mass of $1 + 2 = 3$ (the outgoing particles) is the same as the invariant mass of $4 + 8 = 12$ (the incoming particles). In some circumstances, one may need the incoming *beam* momenta instead. To insert those in place of the incoming parton momenta, use a negative number. In the previous example, the invariant mass of -12 would be the invariant mass of the incoming beams which is equal to the nominal c.m. energy. Of course, there is no difference unless the process is a $2 \to n$ scattering process with structure functions enabled.

Cut window specifications consist of the keyword `within` and two real numbers, optionally followed by additional specifications like

```
cut PT of  4 within 0 150
cut M  of  3 within 80 100 or 180 200 or 500 99999
```

The absence of an upper bound should be marked by a number outside the kinematical range.

4.6 Integration

4.6.1 Running `WHIZARD`

After the input parameters and cuts have been specified, the integration can be started by

Code	Alternative code(s)	# Args	Description
-		$0-2$	No cut
M	Q	1	(Signed) invariant mass $M = \mathrm{sgn}(p^2)\sqrt{\lvert p^2\rvert}$
LM	LQ	1	$\log_{10}\lvert M\rvert$
MSQ	QSQ S T U	1	Squared invariant mass $M^2 = p^2$
E		1	Energy in the lab frame
LE		1	$\log_{10} E$
PT		1	Transverse momentum $p_\perp$
LPT		1	$\log_{10} p_\perp$
PL		1	Longitudinal momentum p_L
P		1	Absolute value of momentum $\lvert\vec{p}\rvert$
Y	RAP RAPIDITY	1	Rapidity y
ETA		1	Pseudorapidity η
DETA	DELTA-ETA	2	Pseudorapidity distance $\Delta\eta$
PH	PHI	1	Azimuthal angle ϕ (lab frame) in radians
PHD	PHID PHI(DEG)	1	Azimuthal angle ϕ (lab frame) in degrees
DPH	DPHI DELTA-PHI	2	Azimuthal distance $\Delta\phi$ (lab frame) in radians
DPHD	DPHID DELTA-PHI(DEG)	2	Azimuthal distance $\Delta\phi$ (lab frame) in degrees
AA	ANGLE-ABS TH-ABS THETA-ABS	1	Absolute polar angle θ_{abs} (lab frame) in radians. Reference axis is the z-axis.
AAD	ANGLE-ABS(DEG)		
	TH-ABS(DEG) THETA-ABS(DEG)	1	Absolute polar angle θ_{abs} (lab frame) in degrees
CTA	COS(TH-ABS) COS(THETA-ABS)	1	$\cos\theta_{\mathrm{abs}}$
A	ANGLE TH THETA	2	Relative polar angle θ (lab frame) in radians
AD	ANGLE(DEG)		
	TH(DEG) THETA(DEG)	2	Relative polar angle θ (lab frame) in degrees
CT	COS(TH) COS(THETA)	2	$\cos\theta$
A*	ANGLE* TH* THETA*	2	Relative polar angle θ^* (rest frame of part.#2) in radians
AD*	ANGLE*(DEG)		
	TH*(DEG) THETA*(DEG)	2	Relative polar angle θ^* (rest frame of part.#2) in degrees
CT*	COS(TH*) COS(THETA*)	2	$\cos\theta^*$
DR	DELTA-R CONE	2	Distance in η-ϕ space, i.e. $\sqrt{\Delta\eta^2 + \Delta\phi^2}$
LDR	LOG-DELTA-R LOG-CONE	2	$\log_{10}\sqrt{\Delta\eta^2 + \Delta\phi^2}$
VBF	VBF_ETA HEMI HEMISPHERE	2	$\eta_1 \times \eta_2$

Table 2: *Cut and histogram code letters. Masses, energies and momenta are measured in GeV.*

```
make run
```

or, if we are in the working directory (`results`),

```
./whizard
```

After each iteration, the result will be displayed on screen and logged in the file `whizard.XXX.out`, where `XXX` is the process tag. An example is shown in Fig. 7. As it is expected, the integration converges towards a stable result, and at the same time the estimated reweighting efficiency increases (allowing for some fluctuations).

The number of iterations and the number of calls per iterations are as given in the input file `whizard.in` (see Fig. 4). Adaptation proceeds in three passes which are separated by a horizontal line in the output:

In the first integration pass, the grids are adapted after each iteration, but the relative weight of the integration channels is kept fixed. In the example of Fig. 7, this pass consists of two iterations with $10,000$ calls each. This pass is useful in particular for many-particle processes, where it takes some time before `VAMP` has found the physical region in each channel.

In the second pass, the grids within each phase space channel and the relative weights of the phase space channels are adapted after each iteration. The example of Fig. 7 shows how the integral converges, the error estimate improves, and the estimated reweighting efficiency increases, until some saturation is reached. Here, the sixth column is of main interest, which is given by the estimator for the relative error multiplied by the square root of the number of calls. One expects a number of the order of unity (or smaller) here for a well-adapted grid, independent of the number of calls. Each time this number improves, it is marked by an asterisk. The best grid will be used later for the final integration and event generation steps.

In the third pass, the relative weights are fixed again, and the remaining iterations are averaged for the final cross-section result.

4.6.2 The phase space configuration

The critical point for the integration is the phase space configuration. This is set up automatically by `WHIZARD` before starting the integration. When integrating a given process for the first time, a phase space configuration is generated and written to the file `whizard.phx`. In subsequent runs for the same process, this file is reused. If you would like to edit and modify this file or if you want to set up your own configuration, the new configuration should be stored in the file `whizard.phs` which takes precedence over `whizard.phx` and will not get overwritten. The automatically generated file `whizard.phx` will be deleted when the process list changes or, in any case, by a `make clean` command.

For each process, `WHIZARD` phase space consists of a set of *channels* which formally correspond to Feynman diagrams. In the automatically generated phase space configuration file `whizard.phx`, these channels are listed in order, using binary codes for the internal lines. The channels that have been generated can be drawn and viewed on screen or printed:

```
make channels
```

This generates a LaTeX file `whizard-channels.tex` which uses the FeynMF package for drawing Feynman graphs, and transforms it into a PostScript file `whizard-channels.ps`. These files are located in the `results` directory. Note that this is currently available only for the automatically generated phase space, not for a phase space setup supplied by the user. (Note further that the Feynman diagrams drawn are only *sample diagrams* for the corresponding phase space channels. Especially if the user edits phase space by removing channels from the file `whizard.phx` by hand, still the full matrix element is taken into account!)

Similarly, the bin distributions of the grids as they have been adapted during the `WHIZARD` run can be converted into histograms by

```
make grids
```

and viewed as the file `whizard-grids.ps` (note that this only works if the user has set the variable `plot_grids_channels` in the `diagnostics_input` block).

Again, it should be kept in mind that the choice of phase space channels does not influence the integration results (*all* Feynman graphs are included in the cross section calculation and event generation, but for the phase space setup only some of them are used — and some are even used twice). However, the accuracy and the speed of convergence depend on this choice. In many cases it will not be necessary to modify anything, but if you are not satisfied with the results of the default configuration, the following remarks may help.

- The channels are grouped in *groves*: Sets of channels that are equivalent in some sense and therefore should get equal weight. All channels within a grove are sampled with the same number of points per iteration which may increase or decrease during the adaptation procedure. To prevent channels from being dropped altogether, one may set `min_calls_per_channel` to a positive value, e.g. 100.

- When a Feynman graph is interpreted as a phase-space parameterization channel, the internal lines are considered either as *resonant*, *log-enhanced*, or *off-shell*. A resonant line either corresponds to a resonance which is kinematically allowed to go on-shell. A log-enhanced line is an (approximately) massless propagator, either in the s-channel (e.g., a photon which splits into massless quarks), or in the t channel (e.g., a Coulomb photon which is exchanged between the initial particles).

- The first criterion for selecting a channel is the number of off-shell lines. The default value is `off_shell_lines=1`. Since in dominant graphs most internal lines are either resonant or log-enhanced, this already covers many cases.

- By this criterion, some important graphs are missed, e.g., the Higgs radiation off a top quark in e^+e^- annihiliation, which has two off-shell lines. The rule is that additional channels are selected with `extra_off_shell_lines` (default 1). For those, the log-enhanced lines count as off-shell. It turns out that this rule is sufficient for essentially all physically relevant cases. However, it may fail in more obscure cases (e.g., if non-resonant regions of phase space are selected by cuts), so increasing these parameters manually, which adds more Feynman diagram classes to the phase-space setup, is a possible option for the user.

- At high energies, multiperipheral graphs with particle exchange become important even though they may be subleading according to the former rules. This fact is accounted for by setting a `threshold_mass_t` value equal to 200 GeV, so W and Z bosons in the t-channel are considered light, and W/Z exchange graphs counted log-enhanced. For the s-channel, the `threshold_mass` value is 50 GeV, so W and Z are considered heavy. Of course, these values may be adjusted if the kinematics is completely different.

- If no phase space channels are found which satisfy all criteria, `WHIZARD` will give it another try with `off_shell_lines` increased.

- Five additional flags give finer control: `single_off_shell_branchings`, `double_off_shell_branchings`, `single_off_shell_decays`, `double_off_shell_decays`, `massive_fsr`. With the exception of the second flag, all are `.true.` by default, so switching them off will remove phase space channels.

- In particularly complicated situations you might wish to have full control and write your own configuration in `whizard.phs`, taking the pre-generated configuration `whizard.phx` as the starting point. To make this easier, for nonzero `extra_off_shell_lines` you will find extra channels in the output which are commented out. Instead of finding the binary coding of Feynman graphs yourself, you may just remove the comment characters for a inactive set of phase space channels.

The number of channels in the default configuration tends to be somewhat too large. Many channels turn out to be irrelevant and will be dropped during the self-adaptation procedure. If you can guess which ones those are, you may use the available parameters and options for manual intervention to optimize the channel selection. As an example, the effective Zee coupling is almost zero because of $\sin^2\theta_w \sim 1/4$, so graphs with Z emission from electron lines are suppressed. You could remove such graphs, either manually, or by removing the Zee vertex from the `whizard.mdl` file before starting the program. (You have to be careful not to drop resonant $Z \to ee$ decays, then.) However, in most cases, while this may save some running time for the adaptation, it is not worth the effort.

On the other hand, it may happen that important channels are missed by the default configuration. In this case, the restrictions should be weakened.

4.6.3 What if it does not converge?

Although `WHIZARD` is able to produce stable results with the default settings for a large variety of scattering processes, it is not possible to guarantee fast convergence of the cross section estimate in all circumstances. There is no simple rule for the achievable accuracy and event generation efficiency for a given process; the examples in Sec. 6 should provide some guideline. *In any case the screen output of the adaptation procedure should be inspected, and questionable results should not be used for simulation.*

Experience shows that in well-behaved $2 \to 4$ processes an accuracy (`Err/Exp` in the output) below 1 and an estimated reweighting efficiency above 10 % is reasonable. Due to the increased

dimensionality of phase space, for $2 \to 6$ processes the numbers are typically worse by a factor 2 to 4.

If convergence is apparently not reached, i.e., there are sizable fluctuations in the result, this may be due to one of the following causes:

- The integral is actually infinite. One should check the cuts if they are sufficient for a finite result. Note that a singularity with a small coefficient is likely to produce an instability rather than an obvious divergence, if the number of calls per iteration is finite.

  A notorious example are processes where the final state contains $\mu^+ e^- \bar{\nu}_e \nu_\mu$ or $\mu^+ \bar{u} d \bar{\nu}_\mu$: For zero muon mass, there is no problem, but if the muon mass is given the physical value, this final state can result from $\mu^+\mu^-$ production and muon decay. The muon width is zero, which is reasonable since the muon decay occurs outside the interaction point, but with cuts on visible particles only, real muon decay makes the integral divergent.

- The number of iterations is insufficient. For simple $2 \to 3$ processes, one reaches a stable result after $2-5$ iterations, but in complicated cases (many particles with photon emission with weak cuts, etc.) $20-30$ iterations may be needed. `WHIZARD` will make a guess, but this may be not sufficient in some cases.

- The number of calls per iteration is insufficient. With increasing dimension and complexity of the phase space, this number should be increased as well.

- The phase space parameterization is inadequate (see above). One may try to include or remove channels by modifying parameters such as the number of off-shell lines allowed in generating parameterizations.

- The parameterization is adequate in principle, but some subdominant (but still important) channels got dropped during the adaptation procedure. This may be prevented by setting `min_calls_per_channel`. The channel weight history can be inspected in the process-specific output file.

- The matrix element is numerically unstable. This is probably not an issue for `O'Mega` or `MadGraph` matrix elements with realistic kinematics, but cannot be excluded. `CompHEP` matrix elements, which result from squared amplitudes, are susceptible for numerical instabilities which occur for extreme kinematics. Actually, if the compiler supports it, one may switch to quadruple precision (configure with `--enable-quadruple`) in order to ameliorate this problem.

- One should be careful to avoid the effects of QED gauge invariance violation, which can be large if unstable particles and photons are around (e.g., single W production in e^+e^- collisions). With `CompHEP` matrix elements, the `gwidth` switch should be set in such cases, while `O'Mega` matrix elements should be generated with the '`f`' option. This enables the *overall factor* scheme, where the resonant contributions to the matrix element are evaluated for zero width of charged particles, and the Breit-Wigner correction factor is applied to the complete matrix element at the end. Another gauge-invariant prescription

(the complex-mass scheme) is foreseen to be implemented in a future version of O'Mega within the context of WHIZARD 2.

4.6.4 Output files

All results are logged in a generic file whizard.out and in process-specific files whizard.XXX.out, where XXX is the process tag. The output files are rewritten after each iteration. Depending on the settings of the diagnostics parameters, they will contain additional information: The contents of the input file are repeated, including also the default values which have been inserted by the program. Furthermore, it shows the detailed history of the individual integration channels and of their relative weights.

After each iteration, the current VAMP grids are written to file as well: whizard.XXX.grc for the current grid and whizard.XXX.grb for the best grid. These may be reused in another run.

```
 ! Begin SLHA data
# SUSY Les Houches Accord Version 1.0
Block MODSEL  # Select model
    1    1    # mSugra
Block SMINPUTS   # Standard Model inputs
    1   1.279340000e+02  # alpha_emˆ(-1)(MZ)
    2   1.166370000e-05  # G_Fermi
    3   1.172000000e-01  # alpha_s(MZ)MSbar
    4   9.118760000e+01  # MZ(pole)
    5   4.250000000e+00  # Mb(mb)
    6   1.743000000e+02  # Mtop(pole)
    7   1.777000000e+00  # Mtau(pole)

⋮

# WHIZARD: The following blocks have not been used:
Block EXTPAR          # non-universal SUSY breaking parameters

⋮

# WHIZARD: The following blocks have been added:
BLOCK PROCESS   # Result of cross section calculation
    1          2         # Number of incoming particles
    2         11         #   Incoming particle: e
    2        -11         #   Incoming particle: a-e
   11          2         # Number of outgoing particles
   12    1000024         #   Outgoing particle: ch1+
   12   -1000024         #   Outgoing particle: a-ch1+
   21    1.62815594E+02  # Cross section in fb
   22    8.00542154E-02  # Cross section error in fb
   23    1.56582525E+00  # Chi-squared value
#
BLOCK CSINFO   # Cross section calculator information
    1    WHIZARD    # Calculator name
    2    1.97       # Calculator version
#
 ! End SLHA data
```

Figure 5: *Sample output in SLHA format*

```
! e-  e+  ->  e-  e+  gamma
! 16   8       1   2    4
process eeg
  cut Q of 10 within -99999 -1
  cut Q of 17 within -99999 -1
  cut M of  3 within 10 99999
  cut E of  4 within  5 99999
  cut PT of 4 within 19 99999
  cut THETA(DEG) of  4 1 within 5 180
  cut THETA(DEG) of  4 2 within 5 180
```

Figure 6: *Cut configuration file*

```
! WHIZARD 1.97 (May 31 2011)
! Reading process data from file whizard.in
! Wrote whizard.out
!
! Process nnbb:
!    e a-e -> nu_e a-nu_e   b a-b
!   32  16 ->    1      2   4   8
....

! WHIZARD run for process nnbb:
!===============================================================================
! It      Calls  Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!-------------------------------------------------------------------------------
! Reading cut configuration data from file whizard.cut1
! No cut data found for process nnbb
! Preparing (fixed weights):   2 samples of      10000 calls ...
   1      20000  5.6761132E+01  1.07E+00    1.88    2.65*  1.45    1.52   2
!-------------------------------------------------------------------------------
! Adapting (variable wgts.):  10 samples of      20000 calls ...
   2      20000  5.9620359E+01  8.15E-01    1.37    1.93*  6.58
   3      20000  5.8344071E+01  4.62E-01    0.79    1.12*  8.99
   4      20000  5.8023577E+01  4.17E-01    0.72    1.02*  9.04
   5      20000  5.7938879E+01  3.83E-01    0.66    0.94* 11.95
   6      20000  5.8152958E+01  3.73E-01    0.64    0.91* 11.85
   7      20000  5.8132996E+01  3.66E-01    0.63    0.89* 12.02
   8      20000  5.8252891E+01  3.50E-01    0.60    0.85* 13.61
   9      20000  5.8296069E+01  3.42E-01    0.59    0.83* 12.74
  10      20000  5.8391405E+01  3.32E-01    0.57    0.81* 15.93
  11      20000  5.8168510E+01  3.19E-01    0.55    0.78* 16.66
!-------------------------------------------------------------------------------
! Integrating (fixed wgts.):   3 samples of      50000 calls ...
  12     150000  5.8232347E+01  1.17E-01    0.20    0.78  10.11    0.49   3
!-------------------------------------------------------------------------------
```

Figure 7: *Sample output of running* ***WHIZARD****.*

4.7 Event generation

4.7.1 Unweighted events

Event generation is activated in the input file, either by specifying a nonzero luminosity

```
&process_input
...
luminosity = 100
/
```

or by a nonzero number of events

```
&simulation_input
n_events = 10000
/
```

In the former case, the actual number of generated events depends on the calculated cross section. If both numbers are given, the one leading to the larger event sample will be used. It is also possible to specify `n_calls`, which is the number of matrix element calls. However, this is more useful for weighted event samples (see below).

The simulation makes sense only if `WHIZARD` could find a stable result for the cross section. This should be judged from inspecting the results displayed on screen and in the log file `whizard.out`.

With event generation activated, `WHIZARD` will not stop after the integration step, but start event generation. By default, `WHIZARD` will generate unweighted events, which result from reweighting the events produced by the `VAMP` sampling algorithm. In the event generation step, importance sampling is used, such that the events are random and have the proper distribution as required for a realistic simulation.

`WHIZARD` is able to write events to file in various formats (note that the manual of `WHIZARD` 2, http://projects.hepforge.org/whizard/manual.pdf), does contain examples for all the different event formats presented here, though the debug format does look different there):

- The file `whizard.evx` contains the events in a binary machine-dependent format. By setting `read_events=T` they can be re-read, bypassing the time-consuming event generation. Writing this file is suppressed if `write_events_raw=F`.

- The file `whizard.evt` (or, alternatively, `whizard.pythia` or `whizard.stdhep`, depending on the format) is written if `write_events=T`. The event file format is controlled by the parameter `write_events_format`. The supported formats are:

  0 Verbose format (ASCII), useful mainly for debugging.

  1 HEPEVT format (ASCII). The format corresponds to the HEPEVT standard, such as if the contents of a HEPEVT common block were written in their natural order (Fig. 8, 9). As an extension, the helicity (-1, 0, or 1) is written for each particle in the same line after `JDAHEP`. For unfragmented events, the first line will contain further information: the number of beam remnants and outgoing partons listed separately, and the event weight.

2 SHORT format (ASCII). This is similar to HEPEVT, but unnecessary entries are suppressed (Fig. 10).

3 LONG format (ASCII). The same as SHORT, but there is an additional line per record which contains the value of the integrand and the integrand ratio (Fig. 10). In a normal run (`recalculate=F`), the ratio is unity (see Sec.5.5).

4 ATHENA format (ASCII). This is the input format used by the ATHENA software (ATLAS). A slight variation of the HEPEVT format above. (cf. Fig. 12)

5 Les Houches Accord format LHA (ASCII). This format implements the Les Houches Accord as in the original `MadGraph` implementation for transferring parton-level events to hadronization packages. To use it, you must set `fragment=T`, and you may set `fragmentation_method=0` since any built-in fragmentation is irrelevant for the output. (cf. Fig. 13)

6 Les Houches Accord format LHEF (ASCII). The new version containing XML headers [18], cf. Fig. 14. For more details cf. the `WHIZARD` 2 manual, http://projects.hepforge.org/whizard/manual.pdf.

11/12/13 PYLIST formats 1,2,3 (ASCII). This is useful in particular for fully fragmented events, but can be used as well if fragmentation is turned off (see the `PYTHIA` manual [1] for details).[26]

20 STDHEP format for writing the HEPEVT common block (binary, machine-independent)

21 STDHEP_UP format for writing the HEPRUP/HEPEUP common blocks (binary, machine-independent)

Note that the formats that require external libraries (`PYTHIA`, `STDHEP`) do not work if the corresponding libraries have not been linked to the main program.

Within each event, the particles will appear in the order determined by the process definition /footnotefor `O'Mega` and `MadGraph`; `CompHEP` may reorder them.. Hadronization packages such as `JETSET` require a particular order of colored particles to properly assign showers, be may necessary for the user to rearrange them before feeding them into the hadronization step. In general, if color-flow information is absent [27], one may set `guess_color_flow` to tell the program that the particles are to be combined as singlets in the order in which they appear in the event record.

Reweighting Monte-Carlo events requires the knowledge of the highest weight. This can never be known with certainty in advance, so there may be some fraction of events with weight greater than one. By default, `WHIZARD` will sum up the excess weight of those events and show the induced error in the total cross section, which should be small compared to the statistical

[26] This works only if `PYTHIA` has been linked. Also it does not work (properly) if `WHIZARD` and the CERN library containing `PYTHIA` have been compiled by different compilers, because `PYTHIA` or `JETSET` cannot connect the proper output unit in that case. So then a file like `fort.<unit>` will appear containing the `PYTHIA/JETSET` events.

[27] For `CompHEP` matrix elements, and for `O'Mega` without the `'c'` option.

```
integer, parameter :: nmxhep = 4000
integer :: nevhep, nhep
integer, dimension(nmxhep) :: isthep, idhep
integer, dimension(2, nmxhep) :: jmohep, jdahep
real(kind=double), dimension(5, nmxhep) :: phep
real(kind=double), dimension(4, nmxhep) :: vhep
common /HEPEVT/ nevhep, nhep, isthep, idhep, &
     & jmohep, jdahep, phep, vhep
```

Figure 8: *Definition of the HEPEVT common block in Fortran 95 format*

error of event generation. If desired, the excess events can be further analyzed by `WHIZARD`'s own analysis capabilities (see below), to make sure that they have no significant impact on final results. By setting the `safety_factor` in the `simulation_input` block to a number greater than one, the amount and weight of excess events (if any) will be reduced. However, the time needed for event generation will be increased by the same factor.

`WHIZARD` will use the adaptation step to determine the highest weight beforehand. This requires sufficient statistics in the adaptation procedure. If many events have to be generated, the default choice for the number of calls per iteration may be too low, and too many excess events are produced. As a rule of thumb, the number of calls per adaptation iteration should be of the order of the number of unweighted events divided by the estimated efficiency.

Alternatively, a "test" run can be made by specifying a nonzero value of `n_events_warmup` in the input file. For instance, one may generate $200,000$ (weighted) events to determine the highest weight before generating the actual sample to be used for the reweighting procedure:

```
&simulation_input
n_events = 10000
n_events_warmup = 200000
/
```

Fortunately, this is not necessary in most cases since the pre-determined highest weight is usually quite accurate.

4.7.2 Weighted events

The default setup will generate unweighted events, which is the proper way of simulating actual physics runs. The cross section error of the simulation run is equal to the square root of the number of events, as it should be.

This requires an unweighting step where, depending on the ratio of the current event's weight and the maximum weight, a large fraction of the generated event sample is thrown away. Clearly, information is lost in this step. To keep all available information, one can use weighted events instead:

```
&simulation_input
unweighted = F
```

```
 write(u, 11) nhep
 do i=1, nhep
    write (u,13) isthep(I), idhep(I), &
         & (jmohep(j,i), j=1,2), (jdahep(j,i), j=1,2)
    write (u,12) (phep(j,i), j=1,5)
    write (u,12) (vhep(j,i), j=1,min(5,ubound(vhep,1)))
 end do

 11 format(10I5)
 12 format(10F17.10)
 13 format(I9,I9,5I5)
```

Figure 9: *Output routine which would produce the event file* ***whizard.evt*** *from the contents of a HEPEVT common block.* ***u*** *is an output unit.*

```
write(u,15) evt%n_out + evt%n_beam_remnants, &
     &      evt%n_out, evt%n_beam_remnants, weight
do i=1, evt%n_out
   p = array(particle_momentum(evt%prt(i)))
   write (u,13) particle_code(evt%prt(i)), evt%hel_out(i)
   write (u,12) p(1:3), p(0), particle_mass(evt%prt(i))
end do
12 format(10F17.10)
13 format(I9,I9,5I5)
15 format(3I5,1X,F17.10)
```

Figure 10: *Output routine for the SHORT event file format. Note that the number of beam remnants and the event weight are not part of the HEPEVT standard. (We here and in the sequel do not include the code for the beam remnants and/or initials.)*

```
write(u,15) evt%n_out + evt%n_beam_remnants, &
     &      evt%n_out, evt%n_beam_remnants, weight
do i=1, evt%n_out
   p = array(particle_momentum(evt%prt(i)))
   write(u,13) particle_code(evt%prt(i)), evt%hel_out(i)
   write(u,12) p(1:3), p(0), particle_mass(evt%prt(i))
end do
write(u, 14) evt%mc_function_value, evt%mc_function_ratio
12 format(10F17.10)
13 format(I9,I9,5I5)
14 format(ES17.10,F17.10)
15 format(3I5,1X,F17.10)
```

Figure 11: *Output routine for the LONG event file format. Note that the number of beam remnants and the event weight are not part of the HEPEVT standard.*

```
write(u,11) evt%count, event_n_entries (evt)
k = 0
do i=1, evt%n_out
   k = k + 1
   p = array(particle_momentum(evt%prt(i)))
   write (u,13) k, 1, particle_code(evt%prt(i)), &
        & evt%n_in+1, 2*evt%n_in, (0, j=1,2)
   write (u,12) p(1:3), p(0), particle_mass(evt%prt(i))
   write (u,12) (0._default, j=1,4)
end do
11 format(10I5)
12 format(10F17.10)
13 format(I9,I9,5I5)
```

Figure 12: *Output routine for the ATHENA event file format. (If the initial particles and/or the beam remnants are to be kept, these are written before the outgoing particles.)*

```
write (u,16) nup, idprup, xwgtup, scalup, aqedup, aqcdup
write (u,17) idup(:nup)
write (u,17) mothup(1,:nup)
write (u,17) mothup(2,:nup)
write (u,17) icolup(1,:nup)
write (u,17) icolup(2,:nup)
write (u,17) istup(:nup)
write (u,17) ispinup(:nup)
do i=1, nup
   write (u,18) i, pup((/4,1,2,3/),i)
end do
16 format(2(1x,I5),1x,F17.10,3(1x,F13.6))
17 format(500(1x,I5))
18 format(1x,I5,4(1x,F17.10))
```

Figure 13: *Output routine for the (old) Les Houches Accord (LHA) event file format.*

```
...
/
```

The number of weighted events is calculated such that (approximately) the same unweighted event sample would result from unweighting them afterwards, as it would from directly generating unweighting events. If `luminosity` or `n_events` do not require a higher number, the total number of weighted events generated is approximately equal to `n_calls`.

If there is more than one process, the luminosity is distributed among them proportional to the individual cross sections. In contrast to unweighted generation, the events are ordered, such that all events belonging to the first process are generated first, then the next one, and so on. The exact number of weighted events for a given process depends on both the cross section and the efficiency. For a process with low efficiency, a larger number of weighted events is needed for the same cross section than for a high-efficiency process. (In unweighted event

samples, the individual event numbers depend only on the cross sections.)

These events can also be written to file and analyzed by an external program. For each event, the event weight is a number between zero and one (if no excess events are present), and it is printed as the last number in the first line of the output. Hence, in this definition the weight is the probability with which the event would have been kept in the sample if an unweighting step had taken place. As a result, differential (binned) distributions should be identical for weighted and unweighted event samples, apart from fluctuations.

Unfortunately, the HEPEVT standard does not allow for event weights. Therefore, the PYLIST and STDHEP output formats which read out the HEPEVT common block cannot be used for weighted events. By contrast, event weights can be transferred using the HEPEUP common block, so the STDHEP_UP format does work for weighted events. Keep in mind that weighted events are ordered, so files containing those must not be truncated or extended.

4.7.3 Built-in analysis

`WHIZARD` has capabilities to analyze the events without reference to external packages. To this end, a configuration file `whizard.cut5` should be set up which has a similar format as `whizard.cut1`. For instance, in Fig. 15 a configuration file is shown which tells `WHIZARD` to analyze the generated event sample in three different ways: to calculate the cross section within a window $80 < M(e^+e^-) < 100$ GeV, then within $180 < M(e^+e^-) < 200$ GeV with an additional cut on $p_\perp(\gamma)$, and finally to generate a histogram of $p_\perp(e^-)$ with a cut on the photon energy.

The histogram specification may take either one of the form

```
histogram PT of 1 within 0 500
histogram PT of 1 within 0 500 nbin 50
histogram PT of 1 within 0 500 step 10
```

Instead of the number of bins (20 by default), the bin width can be specified by the keyword `step`.

The cross section results are displayed on screen. In addition, they are written together with the histogram data to the file `whizard-plots.dat`. If `show_excess=T` has been set in the input file, the summary and the histograms will contain also information about any events with weight greater than one, such that their properties can be examined.

Clearly, for a quick analysis it is not necessary to simulate proper unweighted events. Therefore, one may set `unweighted=F` in order to analyze a sample of weighted events, which saves a lot of execution time.

The contents of `whizard-plots.dat` can be visualized directly by means of the gamelan graphical analysis package which is contained in the distribution. This requires the existence of the `MetaPost` program, which is part of standard LaTeX distributions. Plots can then be made by typing

```
make plots
```

This will result in a Postscript file `whizard-plots.ps` which can be viewed and printed. Note that this works only if you have generated events, otherwise `WHIZARD` might fail with the message

```
No rule to make target plots.
```

The event sample can further be analyzed in terms of color flow, helicity and flavor content. Example

```
! e-  e+  ->  q  qbar  b  bbar
! 16   8      1   2    4   8
process qqbb
  cut M of 3 within 85 95
  color 0 0 4 0 6 0
  helicity  1 -1 -1 1
  helicity -1  1 -1 1
  flavor    2 -2 5 -5
  histogram M of 12 within 0 500
```

This would plot a histogram of the invariant mass of the $b\bar{b}$ pair with the additional requirement that the other quark pair must be $u\bar{u}$, that the b (anti)quarks must be left- resp. right-handed, and that the $u\bar{u}$ and $b\bar{b}$ pairs are each in a color-singlet state.

Color flow is defined in a natural way: The particles are numbered consecutively beginning from the initial ones (do not confuse this with the binary codes used for specifying cuts and histograms). For an incoming quark or gluon, the index of the particle its color flows *to* is given, while for an outgoing quark or gluon it is the index of the particle its color comes *from*. (This differs from WHIZARD's internal convention where incoming and outgoing particles are equally treated, but it is easier to use.) For colorless particles and for antiquarks, the entry is zero. So, in the above example, the q color comes from the $\bar{q}$, and the b color from the $\bar{b}$.

Helicities and flavors are indicated as usual: Fermions and massless vector bosons have helicity ± 1, massive vector bosons can come with helicity 0 in addition, and for scalars the helicity is always zero. Flavor is denoted by standard PDG codes.

The order should be as in the above example: first cut specifications (the event must pass all cuts), then color, helicity and flavor specifications (the event must match any of the given helicity combinations and any of the given flavor combinations), finally histogram specifications. Any of these may be absent; an empty entry will just display the total cross section again. More analysis configurations can be added for the same process, they must be separated by the keyword `and`.

4.8 Fragmentation

The Les Houches standard [9] for transferring events together with weight and color information allows for a consistent interface of partonic event generators and programs that account for showering, fragmentation, and hadronization. In order to use such an external program, WHIZARD has to write events to file in a format that supports this information. Currently, there is the LHA format (code: 5) that writes, for each event, the contents of the HEPEUP block in ASCII format, and the STDHEP_UP format (code: 21) that writes this information into a portable binary STDHEP file. The first record of this file will contain the HEPRUP common block; then, for each event, the HEPEUP common block is written.

Alternatively, WHIZARD provides an internal interface to routines for fragmentation, decays and hadronization. For each event, WHIZARD calls the corresponding external program that fills the HEPEVT common block and writes this to file. To activate this, the parameter `fragment` in the `simulation_input` block must be set to true. Of course, the corresponding library has to be properly linked. Only unweighted events can be fragmented. (This is because the HEPEVT standard does not allow for the event weight to be transferred.)

Once `fragment` is set, various options are available, depending on the value of the parameter `fragmentation_method`:

(0) No fragmentation within WHIZARD (for completeness).

(1) JETSET fragmentation. The final state is fragmented and hadronized by a call to the PYEXEC fragmentation routine inside PYTHIA, i.e., by the former JETSET program. If available, color flow information is used. Otherwise, i.e. for CompHEP matrix elements and O'Mega without the 'c' option, the color connection properties of the final state are guessed by JETSET from the particle ordering, which must therefore carefully be chosen by the user when defining the process. Note that even for processes like $e^+e^- \to jj$ where O'Mega without the option 'c' gives the correct color factors, it only gives the correct color flows for (JETSET) fragmentation when using the 'c' option.

(2) PYTHIA fragmentation. The final state is fragmented and hadronized by making the process generated by WHIZARD an external process within PYTHIA. If possible and depending on the PYTHIA parameter settings, beam remnants are constructed, multiple interactions and pile-up events are added, and more.

(3) This parameter value is intended to transfer control to fragmentation routines written by the user and included in the `user` module. Of course, such code can make use of the HEPRUP and HEPEUP common block contents. For additional flexibility, there is also a parameter `user_fragmentation_mode` (integer) that allows for selecting different versions of user fragmentation.

Full color information is available when using the 'c' option for O'Mega matrix elements (or for MadGraph matrix elements). However, in the other cases, for PYTHIA fragmentation WHIZARD will provide missing color information if can be uniquely determined. Otherwise, it tries to guess color connections from the particle ordering in the event record (just as JETSET does). If color connection information cannot be determined, colored states cannot be fragmented in this way. If fragmentation is nevertheless desired (even though the results may not be accurate), one has to revert to JETSET fragmentation.

No effort is done to streamline WHIZARD and PYTHIA parameters, so changing a value in the WHIZARD input file will have no effect on the fragmentation step. However, PYTHIA parameters can be explicitly set by an interface to PYTHIA's `pygive` routine. For instance, the line

```
pythia_parameters = "MSTP(41)=0; MSTP(81)=0"
```

in the `simulation_input` block disables resonance decays and multiple interactions within PYTHIA. Similarly, the Higgs mass may be set by a call like

```
pythia_parameters = "PMAS(25,1)=115"
```

The detailed list of parameters available is described in the PYTHIA manual [1].

The internal analysis routines described in Sec. 4.7.3 always refer to the event *before* fragmentation. To analyze the fragmented event, the user must use an external package. Note that WHIZARD 2 meanwhile allows for analyzing fragmented events with the internal analysis tools, however.

4.8.1 Parton showers and matching

In WHIZARD there is a version of a k_T-ordered parton shower included written by Sebastian Schmidt. This is, however, a highly experimental version mainly intended for testing purposes. It is quite similar to the PYTHIA parton shower [1], but does contain only very basic tunes to LEP Run 2 jet observables. Furthermore, it is only a shower for final state radiation (FSR); there is no implementation of an initial state shower in WHIZARD 1. The internal shower has been improved a lot and augmented by a different (so-called analytical) parton shower within WHIZARD 2. For more details about the shower algorithm confer the WHIZARD 2 manual http://projects.hepforge.org/whizard/manual.pdf. The internal parton shower can be switched on for event generation in the simulation_input block. Note that the only event format which supports showered events from the WHIZARD-internal parton shower is the (new) Les Houches event format (LHEF, event format 6) as this is the only format that allows for a variable number of particles.

There is, of course, also the possibility to use showering (and hadronization) using the PYTHIA interface. In the default settings, PYTHIA events are constructed with parton showers developing along colored and charged partons. In WHIZARD, fixed-order matrix elements are evaluated. The matching of both is nontrivial. For this reason, PYTHIA parton showers are disabled in the current WHIZARD version. In general, the availability of exact higher-order matrix elements (albeit tree-level) within WHIZARD makes matching without double-counting a delicate task. This problem of potential double counting has been (partially) solved in the framework of so called matching algorithms by either vetoing shower events entering hard matrix elements regimes or reweighting with the corresponding Sudakov factors according to the shower history. None of these algorithms have been implemented within WHIZARD 1, but they are (at least partially) available inside WHIZARD 2.

There is also an experimental Herwig interface, but we do not discuss the details here.

4.8.2 Mixing WHIZARD and PYTHIA events

Since WHIZARD provides an external process to PYTHIA when PYTHIA fragmentation is enabled, it is perfectly possible to mix PYTHIA and WHIZARD events. To this end, one should specify PYTHIA processes in the simulation_input block. For instance,

```
pythia_processes = "22, 24"
```

will simulate $e^+e^- \to ZH$ and ZZ in addition to the WHIZARD-generated processes. There are some caveats, however:

- PYTHIA processes are generated only if PYTHIA fragmentation is enabled. Consequently, the PYTHIA events are mixed with the *fully fragmented* WHIZARD events and optionally written to file. There is no way to use the built-in analysis for those events.
- Since parton showers (ISR and FSR) are disabled globally, this applies to the PYTHIA processes as well. However, one could switch the showering on by modifying the inputs.
- When the number of events to generate is derived from the luminosity input parameter, it is assumed that no PYTHIA events are present. To obtain the actual luminosity corresponding to an event sample in the presence of PYTHIA events, this number has to be multiplied by the fraction of WHIZARD events in this sample. The parameter n_events always refers to the total number of events. When show_pythia_statistics is true, a summary of the generated channels and cross sections is printed.
- One should be careful not to double-count processes, except if this is actually desired (e.g., for comparing WHIZARD and PYTHIA event samples for related processes).

4.8.3 PYTHIA event display

When fragmentation is enabled, the conventions for writing events to file are changed. The verbose format is then not available since some information (used internally by WHIZARD) is lost after fragmentation. The format codes 1 (HEPEVT), 2 (SHORT), 3 (LONG), and 20/21 (STDHEP) are available. In addition, there are three new format codes (11, 12, 13) which allow for displaying events using PYTHIA's PYLIST call. The respective argument given to PYLIST is 1, 2, or 3, corresponding to three different verbosity levels. (For automatic processing, the more condensed HEPEVT or STDHEP formats are better suited.) Note again that these output formats might not work properly if you use a CERN library compiled with a different compiler than the WHIZARD core program, because in this case the output units of the two compiler libraries are independent of each other. The PYTHIA events then just appear in a file fort.<unit>, where <unit> is the current output unit.

4.9 Controlling WHIZARD runs

For a new process, it is not easy to predict how long it will take to reach a sufficiently precise result or to generate a sufficiently large event sample. The achievable efficiency and accuracy and the overall running time strongly depend on the complexity of the process, the singularity structure of the matrix element and how well WHIZARD is able to map it, the presence or absence of structure functions, and, clearly, the evaluation time of the matrix element for a single event.

Ideally, computer time and space is abundant, one calcaulates cross sections to the highest possible accuracy, and infinitely large event samples are generated. In practice, one does not want to wait too long for useable results, and one is limited by CPU time and space restrictions. Priorities have to be defined. To meet these needs, various switches and levers are available that control WHIZARD runs. All of them are set in the main input file and are listed in the tables of Sec. 4.4.

4.9.1 Grid adaptation

Parameters that control the grid adaptation and integration passes of `WHIZARD` are collected in the `integration_input` block in the input file, listed in Sec. 4.4.3

Obviously, the array `calls` is the principal place for improving the accuracy and efficiency of the adaptation and integration runs. In no case more iterations or calls than specified in `calls` will be executed, where `WHIZARD` uses default values for entries left empty by the user.

Increasing the number of adaptation iterations normally improves (decreases) the constant α in the error estimate for the total cross section,

$$\frac{\Delta\sigma}{\sigma} = \frac{\alpha}{\sqrt{N_{\text{events}}}}, \tag{7}$$

until a limit is reached which depends on the matrix element singularity structure and on the choice of phase space integration channels. The estimate for α is printed as the *accuracy* in the screen output table. Increasing the number of points per iteration tends to decrease the number of iterations needed to reach this limit. In particular, this improves the precision in estimating α and reduces inevitable fluctuations around the ideal grid configuration. Simultaneously the reweighting *efficiency* ϵ increases, which is the ratio of the average to the maximum event weight. Note that fluctuations in the ϵ estimate are significantly stronger than for α.

If the program has been killed for any reason, it can be restarted with `read_grids` set, so adaptation is resumed after the last complete iteration. This mechanism can also be used to increase the number of iterations afterwards for a job that has already been completed.

However, one wants to avoid a unnecessary large number of iterations and calls. Adaptation iterations are skipped if any one of the following parameters is set:

`accuracy_goal`: If this number is greater than zero, adaptation will stop when the accuracy estimate of the last iteration is smaller.

`efficiency_goal`: If this number is greater than zero, adaptation will stop when the efficiency estimate of the last iteration is higher. Note that the efficiency is measured in percent.

`time_limit_adaptation`: If this number is greater than zero, adaptation will stop when the real time after the last iteration is later than the specified time limit, measured in minutes from the program start.

In any case, the program will not stop but the third pass (integration) will be executed after the adaptation pass, where the number of integration steps and calls per iteration are taken from the corresponding `calls` entry.

4.9.2 Simulation

The reweighting efficiency in the final event generation pass is determined by the grid quality, measured by the efficiency ϵ. Since the precision in estimating ϵ is not very high, it may be appropriate to insert a `safety_factor` $f > 1$ to avoid excess events with weight > 1. The time needed for generating a fixed number of unweighted events is then proportional to $1/(f\epsilon)$. As

a rule, investing more time in grid adaptation results in a better ϵ value, and thus less time for event generation, up to a certain limit.

The number of unweighted events is specified by either setting the parameter `n_events` in the `simulation_input` block, or by a `luminosity` value in the `process_input` block. The number that leads to a larger event sample will take priority.

If the program is killed during event generation, the event sample is incomplete. The switch `catch_signals` can be used to catch terminate signals if the OS and compiler supports it, so it is guaranteed the the event files are properly closed before the program terminates. Then, `WHIZARD` can be restarted with `read_events` set (which implies `read_grids`). The events already generated will be reread from the raw event file `whizard.evx` until the end is reached, and further events will be appended to this file. Event output files in other formats are regenerated on the fly. Thisy method can also be used to increase an event sample afterwards, spending CPU time essentially on newly generated events only.

4.9.3 Hard limits

There are situations where one has to control the amount of memory, CPU time, and disk space which the program uses. In particular, batch installations put limits on these resources and kill processes which exceed them.

Some of these issues can be handled by `WHIZARD`:

`time_limit`: This parameter (in minutes) sets an internal limit on the wallclock time the run may use. When it is exceeded, the program terminates. Depending whether the termination occured during integration or simulation, the program can be restarted such that it will resume after the last grid or event written, respectively.

`events_per_file`: This parameter takes care of event files becoming too large: once it is exceeded, a new file is begun. An external analysis would then process the files consecutively. Note that this does not apply to the internal event file `whizard.evx` which cannot be split; if this file becomes too large, one has to switch it off by `write_events_raw=F`.

`bytes_per_file`: The analogous parameter which counts the number of bytes allowed for an event file. This relies on certain assumptions on the number of bytes an event takes in a particular format.

```
subroutine lhef_write_init (unit, process_id)
  integer, intent(in) :: unit
  integer :: i
  character(*), dimension(:), intent(in) :: process_id
  write (unit, "(A)")  '<LesHouchesEvents version="1.0">'
  write (unit, "(A)")  '<header>'
  write (unit, "(2x,A)")  '<generator>'
  write (unit, "(4x,A)")  &
       '<name version="<<Version>>" date="<<Date>>">' // &
       'WHIZARD</name>'
  do i = 1, size (process_id)
     write (unit, "(4x,A)") &
          '<process_id>' // trim (adjustl (process_id(i))) &
          // '</process_id>'
  end do
  write (unit, "(2x,A)")  '</generator>'
  write (unit, "(A)")  '</header>'
end subroutine lhef_write_init

subroutine lhef_write_heprup (unit, process_id)
  character(*), dimension(:), intent(in) :: process_id
  integer, intent(in) :: unit
  integer :: i
  write (unit, "(A)") '<init>'
  write (unit, "(2(2x,I10),2(2x,G22.15E3),8x,6(2x,I10))") &
       idbmup(1:2), ebmup(1:2), &
       pdfgup(1:2), pdfsup(1:2), &
       idwtup, nprup
  do i = 1, size (process_id)
     write (unit, "(3(2x,G22.15E3),8x,1(2x,I10))") &
          xsecup(i), xerrup(i), xmaxup(i), lprup(i)
  end do
  write (unit, "(A)") '</init>'
end subroutine lhef_write_heprup

subroutine lhef_write_hepeup (unit, pro_id)
  integer, intent(in) :: unit, pro_id
  integer :: i
  idprup = pro_id
  write (unit, "(A)") '<event>'
  write (unit, "(2(2x,I10),2(2x,G22.15E3),8x,2(2x,G22.15E3))") &
       nup, idprup, xwgtup, scalup, aqedup, aqcdup
  do i = 1, nup
     write (unit, "(6(2x,I10),8x,3(2x,G22.15E3),8x,2(2x,G22.15E3),32x,2(2x,G22.15E3))") &
          idup(i), istup(i), &
          mothup(1:2,i), icolup(1:2,i), &
          pup(1:5,i), vtimup(i), spinup(i)
  end do
  write (unit, "(A)") '</event>'
end subroutine lhef_write_hepeup

subroutine lhef_write_final (unit)
  integer, intent(in) :: unit
  write (unit, "(A)")  '</LesHouchesEvents>'
end subroutine lhef_write_final
```

Figure 14: *Output routine for the (new) Les Houches Accord (LHEF) event file format.*

```
! e-  e+  ->  e-  e+  gamma
! 16   8       1   2    4
process eeg
  cut M of 3 within  80 100
and
  cut M  of 3 within 180 200
  cut PT of 4 within 100 99999
and
  cut E of 4 within 0 100
  histogram PT of 1 within 0 500
```

Figure 15: *Analysis configuration file.*

5 Advanced usage of WHIZARD

In the preceding chapters we have described the use of WHIZARD as a stand-alone program. As such, it is able to calculate total cross sections and to generate events on the partonic level, or even on the hadronic level if fragmentation is enabled. The events can be analyzed online (on the partonic level) or written to file (on the partonic or on the hadronic level).

This already covers many real-life applications of a Monte-Carlo integration and event-generation package. For the theorist, it may be sufficient to calculate a total cross section with a given set of parameters and to plot differential distributions. For an experimental simulation, the generated event files can be used for further processing.

However, one encounters situations where a more elaborate framework is desired. For instance, one would like to consider multiple sets of cuts, energies and parameter sets. For each set, the input file has to be modified, the program has to be run, and the results have to be stored somewhere in order to be compared later. This can easily become cumbersome, and it is desirable to have a generic approach. Another common problem is the selection of a certain subclass of Feynman diagrams contributing to a given process, either for speeding up the calculation by dropping irrelevant terms, or for getting a better understanding of the relative weight of interfering subprocesses. Some possibilities are described in the following sections.

5.1 Feynman diagram selections

While the matrix element generators called by WHIZARD, by default, generate an amplitude which corresponds to the complete set of Feynman diagrams which connect a given initial and final state, there are problems where only a subset is needed. Therefore, WHIZARD has support for diagram selections. Since O'Mega does not use Feynman diagrams, this option is available only when using the CompHEP and MadGraph matrix element generators. (O'Mega, however, supports restrictions on the internal lines of an amplitude. This is explained below in Sec. 5.2.) Since the CompHEP and MadGraph matrix element generators are considered to be deprecated features we describe Feynman diagram selection here only briefly.

The selection must be made before the final executable is generated. In the configuration file, the process must be marked accordingly by the letter "s" in the last column, for instance:

```
# Processes
# Tag           In      Out             Method  Option
#=================================================
cc3             e1,E1   e2,N2,u,D       chep    s
```

Note that this makes sense only for the methods chep and mad, as mentioned before. Next, the diagram numbers to be selected have to be determined. To this end, for each process with this option a diagram selection file has to be put into the configuration directory.

Since the user usually does not know diagram numbers in advance, it is possible to do an empty run of the matrix element generators first. If the command

```
make diagrams
```

is issued, matrix element generation is started as usual, but the Fortran95 source code is not actually generated (at least, with CompHEP: for the other generators it makes little difference). Instead, after this command is completed, you will find PostScript files with Feynman diagrams in graphical representation in the subdirectory

processes-src. View or print them and note, for each process, the diagrams you are interested in. In the subdirectory conf you should now find, for each process, a file with the extension .sel, which contains one entry for each diagram. Initially, all entries are set 0 (false). As a consequence, if the file is not modified, a second WHIZARD run would fail since all diagrams are deleted initially. Change all entries you wish to enable into T or 1 (true). If you now rerun WHIZARD to generate the actual source code (you might need to touch the file whizard.prc in the conf subdirectory in order to force re-generation of the process under consideration)

```
make proc
```

or even

```
make prg install
```

WHIZARD should read the selection files and take into account only the specified subsets of diagrams. You may control this by looking at the PostScript output this second pass will produce: it should contain only the Feynman diagrams you have selected.

Needless to say, this feature should be used with great care. By violating electroweak or even electromagnetic gauge invariance, the results may be off by far, since delicate cancellations may have been spoiled. As a rule, never delete diagrams in any gauge other than unitarity gauge (in the CompHEP) case, and remember which diagrams have been selected (i.e., save the PostScript file somewhere) when storing results, since the output of the WHIZARD executable itself does not contain any hint that only a subset of Feynman diagrams have been taken into account. In many cases the same effect can also be achieved in a simpler way by putting some parameter to extreme values. For instance, instead of removing Higgs diagrams by hand, one may set the Higgs mass to a very large value — the results will then be valid in any gauge.

5.2 Restrictions on intermediate states

As an alternative (and often more convenient) method for selecting Feynman diagrams, the O'Mega interface supports restrictions on internal lines.[28] As an example, the following specification selects the resonant WW diagrams, analogous to the previous example:

```
# Processes
# Tag           In      Out             Method  Option
#=================================================
cc3             e1,E1   e1,N1,u,D       omega   r: 3+4~W-&&5+6~W+
```

Simultaneous requirements for a process are concatenated by && (AND). Note that spaces in the restrictions option ('r') might lead to a failure of the scripts triggering the O'Mega matrix element generation.

The following examples select only diagrams with an s-channel and t-channel γ^*, respectively:

```
cc_gamma_s      e1,E1   e1,N1,u,D       omega   r: 3+4+5+6~A
cc_gamma_t      e1,E1   e1,N1,u,D       omega   r: 1+3~A
```

The in- and out-particles are numbered consecutively. The tilde selects only diagrams where the specified combination of momenta corresponds to the propagator for the given particle. A question mark instead of a particle name stands for all possible propagators with the specified

[28] Similar restrictions are possible with the official CompHEP and MadGraph versions, but not supported by WHIZARD.

momentum combination. *These particle names have to follow the O'Mega notation; they are not translated by WHIZARD as in the process specification.* For colored intermediate states, WHIZARD takes care automatically that the intermediate particle can appear in different color flow combinations (in WHIZARD 2 this is handled directly by O'Mega itself in the matrix element generation). It is important to keep in mind that such a selection typically violates gauge invariance, since the intermediate particle is not forced on shell. (An on-shell option is also present in O'Mega, but not yet supported by WHIZARD.)

The || (OR) relation is also possible, and sometimes needed, as in this example:

```
ww_or_zz   e1,E1   e1,N1,E1,n1    omega   r: 3+4~W- && 5+6~W+ || 3+5~Z && 4+6~Z
```

AND and OR relations can be grouped by parantheses (). Note that combinations of AND and OR as implemented in the setup of O'Mega in WHIZARD 1 could lead to logical inconsistencies in the selections of intermediate states, so use this feature with great care. In WHIZARD 2 this has therefore been replaced by the option to use lists of intermediate states.

Currently, WHIZARD does not interpret or use such restrictions for its phase space setup, so the possible speedup is constrained to the matrix element evaluation. Nevertheless, this feature can be very useful. Such an option is foreseen for a future release of WHIZARD 2, however.

5.3 User-defined spectra and structure functions

It is possible to insert arbitrary beam spectra into the WHIZARD code. If the corresponding code is included in the user module, it can be called from the main program. To do this, in the whizard-src subdirectory, copy the file user.f90.default to user.f90 and edit the appropriate functions defined within this module according to your needs. The code you have written will be compiled instead of the default code.

The comments in the user.f90.default source should guide you. There is the possibility to define either single-beam (factorized) or correlated spectra. The particle codes which have been specified in the beam_input block of the input file can be accessed in order to select between different spectra for different beams.

In the input file, for each beam there is a master switch USER_spectrum_on which determines whether the user code is called. If no user code has been written, by default a uniform x spectrum between 0 and 1 is inserted which could be used for testing purposes. Furthermore, two parameters can be specified: USER_spectrum_mode, an integer which the user function can access to select between different spectra for the same particle, and USER_spectrum_sqrts which by default is the total c.m. energy, but in principle can be used to transfer any real constant parameter.

Apart from user-defined spectra, user-defined structure functions can be inserted. They are handled in an exactly analogous way (with spectrum replaced by strfun in parameter names). The only difference is that user spectra are applied *before* any built-in spectra and structure functions (like beamstrahlung, PDFs, etc.), while user structure functions are applied *after* built in ones, i.e., they are called for the partonic initial state. For applications, think of a photon collider spectrum for the initial beams as opposed to the scattering of W bosons emitted from quarks.

The efficiency of the program in the presence of user-defined spectra or structure functions depends on the way they are implemented by the user. It is straightforward to insert the corresponding functions in a literal way as functions $f(x)$ where x is the beam energy fraction. However, if f is strongly peaked this might lead to poor performance. In many cases one rather should apply a variable transformation

$$\int_0^1 dx\, f(x)\, \sigma(xs) = \int_0^1 dy\, \frac{dg}{dy}\, f(g(y))\, \sigma(xs) \tag{8}$$

where $x = g(y)$. The user interface allows for this, where the input parameters are identified with y and the output parameters $x = g(y)$ can be different. In that case, the spectral function $f(x) = f(g(y))$ has to be multiplied by the Jacobian dg/dy within the structure function code.

The implementation can be checked by making use of the "test" matrix elements (Sec. 4.1.2). You may set up a placeholder process like

```
sftest   e1,E1   A,A    test
```

where the final state is massless and remove all cuts (in the example, set `default_Q_cut=0` or define an empty entry in the cut file). The matrix element is set constant, and the would-be cross section for the process is the integral of the structure function

$$\sigma = \int_0^1 dx\, f(x) \tag{9}$$

which is usually known. Furthermore, the energy distribution of the final state should reproduce the structure function shape.

In the current implementation, beam polarization (if any) cannot be accessed or modified by the user spectra or structure functions. Beam polarization will be kept unchanged if the in- and out-particle of the corresponding splitting coincide (as for ISR), and it will be ignored in the opposite case (as for PDFs).

5.4 User-defined cuts

In many applications the built-in capabilities for cuts are not sufficient. If cuts are defined using the standard interface, all cuts are composed such that an event is only retained if it passes all cuts (logical *and*). There can be several windows in one cut variable, but beyond that it is not possible to compose cuts by a logical *or*. Furthermore, you may be missing your favorite cut model in the list of predefined variables. Note that `WHIZARD` 2 revised this model and makes the definition of almost arbitrary functions for cuts on (combinations of) particles possible.

User-defined cut algorithms can easily be placed in the `user` module which comes with the distribution, the same module where user-defined spectra can be coded. To activate the extra code, the parameter `user_cut_mode` in the input file should be set to a nonzero integer. Then, the new cut function will be applied throughout program execution in addition to the cut conditions defined in the standard way. Within the cut function, the numerical value of `user_cut_mode` is used to select one of several cut modes defined by the user. (To disable the default cuts, which may be desired in this case, write an empty process entry in `whizard.cut1`.)

The `cut` function takes an array of momenta (this type and the functions operating on it is defined in the module `lorentz.f90`) and results in the logical variable `accept`. The default behavior is to unconditionally set `accept=.true.`, but user code would change that, deciding whether to accept the event based on the momentum configuration and the particle codes.

As an example, let us implement a cut such that an event is accepted if there is a photon with energy larger than 50 GeV. (This is not possible by the built-in cut capabilities, since for a generic process we do not know which photon to check.) To this end, we first make a copy of the standard (no-op) user file, which then can be changed:

```
cd whizard-src
cp user.f90.default user.f90
```

The file `user.f90` then has to be edited. We take the section which refers to the user cut definition, which originally reads

```
! This minimal cut function accepts everything

  function cut (p, code, mode) result (accept)
    type(four_momentum), dimension(:), intent(in) :: p
    integer, dimension(:), intent(in) :: code
    integer, intent(in) :: mode
    logical :: accept
    select case (mode)
    case default
       accept = .true.
    end select
  end function cut
```

and change it as follows:

```
    ! This cut function accepts an event only if there is a photon with
!   energy larger than 50 GeV

  function cut (p, code, mode) result (accept)
    type(four_momentum), dimension(:), intent(in) :: p
    integer, dimension(:), intent(in) :: code
    integer, intent(in) :: mode
    logical :: accept
    integer :: i
    select case (mode)
    case (1)
       accept = .false.
       do i = 1, size (code)
          if (code(i) == PHOTON .and. energy (p(i)) > 50) then
             accept = .true.
          end if
       end do
    case default
       accept = .true.
    end select
  end function cut
```

We have inserted a loop over all outgoing particles, such that the code can be applied to an arbitrary process. For each particle `i`, its PDG code is checked against the predefined code `PHOTON` (which is defined as 22 in `particles.f90`), and the function `energy` is evaluated for the corresponding momentum `p(i)`. This function, which takes a variable of type `four_momentum` as argument and returns a real number of default kind (i.e., double precision), is defined in `lorentz.f90`.

When `WHIZARD` is now (re-)compiled and installed,

```
make whizard install
```

the user code becomes part of the program. To activate it, the `integration_input` in the `whizard.in` file has to specify a nonzero user-cut mode, i.e.,

```
&integration_input
...
user_cut_mode = 1
/
```

since in the function defined above, the `mode` 1 refers to the special treatment of photons. When the program is started with this setting, you will see the message

```
! Activating additional preselection cuts in user.f90: Mode # 1
```

in the screen output, and the cross section and event distribution should reflect the new cut condition.

5.5 Reweighting matrix elements

Beyond the possibility to apply arbitrary cuts, the user might be interested in reweighting events. One should distinguish two different situations:

1. A given event sample is rescanned, recalculating the matrix element values event by event with some modification applied. The resulting events carry a weight which is given by the ratio of the new matrix element values compared to the old ones. The event selection is still based on the unmodified values.

   To achieve this, one should first generate a reference sample in the usual way (adaptation and event generation) which results in the raw event file `whizard.evx`. Then, one may change the input parameters as desired, set the parameter `recalculate=T`, and start a second run. This will have two effects: First, the parameters `read_grids` and `read_events` are assumed also to be set (even if the user has not modified them), together with `read_grids_force` and `read_events_force`.[29] Thus, the grids and events of the first run are reused. Second, when the event sample is read, the particle momenta

[29] The latter are necessary because the procedure makes sense only if input data have changed, so the checksum will be invalid. This makes the user responsible for guaranteeing compatible input data in both runs; otherwise the program may crash.

are left unchanged, but the matrix element values and the quantum number probabilities are recalculated, using the new input data.

The change in the matrix element values will be taken into account in the internal analysis: The integral and histograms will be calculated using the ratio of new and old matrix element values as event weights. If the results of the reference run have been stored (one should rename output files to prevent them from being overwritten), the results can easily be compared.

To make further use of the reweighted events, one can write the events to file (using the option `write_events=T`) in "long" (ASCII) format: `write_events_format=3`. In addition to the particle codes, helicities and momenta, the file will contain the (recalculated) matrix element value and the ratio of new and old values for each event. Again, if several parameter sets are to be compared, one should make sure to rename the output file `whizard.evt` after each run to prevent overwriting.

In any case, one has to be careful. While it does not harm to change a coupling constant by a small amount (e.g., an anomalous coupling) and rescan the events, modifying particle masses and cuts may have undesired side-effects since the momentum configurations are left intact. For instance, with new mass assignments particles may no longer be on-shell, such that delicate gauge cancellations in the matrix elements are invalid, resulting in unphysical effects.

2. The matrix element function itself is modified by a user-defined weight, a function of the outgoing momenta. The weight function is applied throughout adaptation and event generation, so the integral and the event selection depend on it.

   A natural application is the calculation of integrated observables other than the total cross section, e.g. moments of the angular distribution. One might also think of radiative corrections in "K-factor" form: a function of the momenta which multiplies the lowest-order matrix element.[30] The corresponding code can be inserted in the `weight` routine in the `user.f90` module. Similar to the `cut` routine (Sec.5.4), it takes the array of particle momenta and particle codes to generate a single number `weight`. In the default routine, `weight` is always equal to one. The user may change this to an arbitrary function, provided it is nonnegative in the whole accessible phase space.[31] The user-defined weight function is then activated by setting `user_weight_mode` nonzero in the input file. The numerical value can be used to select between different reweighting modes.

It is perfectly possible to combine the two approaches: One may define one or more reweighting functions in the `user` module. As a reference run, events are generated with

[30] NLO matrix elements are nontrivial, and `WHIZARD` may be not the appropriate program for dealing with them. An obvious limitation is that the user weight function does not take polarization into account. However, several NLO matrix elements have been interfaced to `WHIZARD`, and in `WHIZARD` 2 an automatic generation of NLO infrastructure is being prepared.

[31] If used with `recalculate=T`, this limitation is absent, since reweighting will be applied *after* adaptation and event rejection. In that case, an unmodified event sample has to be generated first. Furthermore, in [19,20] integrations of indefinite function have been performed with `WHIZARD`.

user_weight_mode=0. Then, both recalculate is set and the mode is set nonzero in a second run. A user-defined cut function can be treated analogously.

5.6 Command-line options

WHIZARD allows to specify options on the command line. Options are available both in long form (double dashes) and in short form (a single dash). For instance, the following command executes WHIZARD in subdirectory tmp for the process ee_nnh, without reading an input file:

```
./whizard -nd tmp --process_input 'process_id = "ee_nnh"'
```

5.6.1 General options

These options are executed by WHIZARD *before* any files are read or written

Option	Long version	Value	Description
-h	--help		Display usage message and exit.
-V	--version		Display version string and exit.
-l	--list		List all available processes and exit.
-d	--directory	*directory*	Set working directory.
-f	--filename	*file name*	Set default base name for all input and output files (without extension).
-i	--input_file	*file name*	Set base name for the main input file (without extension). This overrides the -f option.
-n	--no_input_file		Do not read the main input file.

If a working directory different from the default one is chosen by the -d option, one has to make sure that either a phase space file (.phs or .phx) or the model definition file (.mdl) is present in the chosen directory. Otherwise, one could use command-line options to point to files in the initial directory (or anywhere else), for instance:

```
./whizard --directory tmp --integration_input 'read_model_file="./whizard"'
```

When a --directory setting is in effect, it can be overriden by filenames that begin with "/" (absolute file name) or with "./" (filename relative to the initial directory). Therefore, in the example the model file whizard.mdl will be looked for in the directory where the command was executed, while all other files will be looked for in or written to the subdirectory tmp.

5.6.2 Options for setting parameters

Parameters can be set not just in the input file, but also on the command line. The command-line parameters are read *after* all input files have been read; they override settings in the input file(s). Thus, one can set up a generic input file whizard.in where the constant parameters are defined (or no input file at all) and use the command line to run WHIZARD for, e.g., a specific process, beam energy and Higgs boson mass:

```
./whizard -p 'process_id="ee_nnh" sqrts=800' -P 'mH = 130'
```

Note that the option arguments are interpreted verbatim as if they were contained in the input file, so string values have to be enclosed by extra quotes.

Option	Long version	Value	Description
`-p`	`--process_input`	*parameters*	set parameters in the `&process_input` block.
`-I`	`--integration_input`	*parameters*	set parameters in the `&integration_input` block.
`-S`	`--simulation_input`	*parameters*	set parameters in the `&simulation_input` block.
`-D`	`--diagnostics_input`	*parameters*	set parameters in the `&diagnostics_input` block.
`-P`	`--parameter_input`	*parameters*	set parameters in the `&parameter_input` block.
`-B1`	`--beam_input1`	*parameters*	set parameters in the first `&beam_input` block.
`-B2`	`--beam_input2`	*parameters*	set parameters in the second `&beam_input` block.

5.7 Tables and file management

The `WHIZARD` program, after it has been compiled with a specific list of processes, may be viewed as a kind of filter: It takes as input

- the model definition file `whizard.mdl`,
- the parameter definition file `whizard.in`, and optionally
- the cut definition file `whizard.cut1`
- and the cut/analysis definition file `whizard.cut5`,

and transforms the information contained within into output files, namely

- the integration results and diagnostics, collected in the file `whizard.out`,
- the integration grids, usable to bypass further time-consuming adaptation runs (`whizard.grb`), and optionally
- an event file usable for re-running the program (`whizard.evx`);
- an event file for further processing by external programs (`whizard.evt`);
- and a set of histograms (`whizard-plots.dat`) which can be converted to PostScript format (`whizard-plots.ps`), or alternatively read in by an analysis package like `ROOT`.

For a second WHIZARD run, the input files have to be edited, and the output files will be overwritten if they have not been moved to a different location. Instead of doing this by hand, one could write a script that executes a loop where in each iteration the relevant line in the input file is modified, the program is run, and the output files are renamed or moved such that they do not get overwritten. Here is such a script:

```
#!/usr/bin/perl
# Loop over the total energy, write a table and store the results
# in subdirectories

$sqrts_min = 350;  $sqrts_max = 800;  $sqrts_step = 50;

# These files will be preserved
@outfiles = ("whizard.out", "whizard.grb", "whizard.evt");

# This will become the table of results
$resultfile = "whizard.out";
$table = sprintf(" %-10s  %-20s  %-20s\n", "sqrts [GeV]",
                 "Total cross section [fb]", "Integration error [fb]");

# Main loop
for ($sqrts=$sqrts_min; $sqrts<=$sqrts_max; $sqrts=$sqrts+$sqrts_step) {

    # Read the input file template, modify the sqrts entry, and write the
    # new input file
    $infile = "whizard.in.0";  $outfile = "whizard.in";
    open(INFILE, $infile) or die "Can't open $infile for reading";
    open(OUTFILE, ">$outfile") or die "Can't open $outfile for writing";
    while (<INFILE>) {
        s/sqrts = \d+/sqrts = $sqrts/;
        print OUTFILE;
    }
    close(OUTFILE);
    close(INFILE);

    # Run WHIZARD
    print "\n**** sqrts = $sqrts ****\n";
    $stat = system("./whizard");
    $stat==0 or die "WHIZARD run exited abnormally";

    # Extract the integration results and add them to the table
    open(RESULTS, "grep 'integral =' $resultfile|")
        or die "Grep failed on $resultfile";
    while(<RESULTS>) { ($dummy, $equals, $integral) = split; }
    close(RESULTS);
    open(RESULTS, "grep 'error    =' $resultfile|")
        or die "Grep failed on $resultfile";
    while(<RESULTS>) { ($dummy, $equals, $error) = split; }
    close(RESULTS);
    $table .= sprintf(" %10g  %20g  %20g\n",
                      $sqrts, $integral, $error);
```

```
    # Create a subdirectory and move the results there
    $dirname = "sqrts=$sqrts";
    mkdir($dirname, oct("0755")) or die "Can't mkdir $dirname";
    foreach $file(@outfiles) {
        system("mv", $file, $dirname);
    }

}

print "\n*** Results:\n";
print $table;

print "*** End of WHIZARD runs\n";
exit 0;
```

The script assumes that the subdirectories do not yet exist, and that you have a template `whizard.in.0` for the input file which contains an entry like `sqrts = 0`. The table is shown on screen after all runs have been completed.

This kind of task can be formalized further. The program `whizwrap` takes care of loops over parameters and provides a convenient management of output files. It is called from the command line; the kind of task to be performed by `WHIZARD` is specified in terms of command-line options. Note that within `WHIZARD` 2 these tasks can be easily steered from the input file by means of its powerful interpreter language SINDARIN. Cf. the manual of `WHIZARD` 2 for the details.

5.8 `WHIZARD` as a subroutine library

A more ambitious enterprise is the integration of `WHIZARD` into a larger framework which may also account for the subsequent steps of experimental simulation and analysis. Even if this is not desired, it may be useful to access `WHIZARD` by subroutine calls in wrapper programs written in Fortran, C, or other program languages.

To make this possible, `WHIZARD` is organized as a set of libraries (in the UNIX sense). When the configuration file (the process list) has been specified, instead of making a stand-alone executable by

```
make prg
```

one may generate the libraries only

```
make libs
```

which then can be linked with a user-supplied main program.

To facilitate this procedure, after executing `make libs`, the user will find the generated libraries in the subdirectory `lib` (except for system-supplied libraries from the CERNLIB), together with a script called `whizard.ld` in the `bin` subdirectory which links an arbitrary

program with the libraries in correct order. Thus, the user may write his own Fortran 95 main program, call it, e.g., `my_main.f90`, compile it[32] and execute

```
bin/whizard.ld -o my_whizard my_main.o
```

to generate his own version of `WHIZARD`.[33]

Here is a main program that performs the same task as the script shown in the preceding section:

```
! Loop over the total energy, write a table and store the results in
! separate output files.
program my_main

  use kinds       ! Defines the default real kind
  use whizard     ! Defines the WHIZARD subroutines
  implicit none   ! No implicit typing

  ! Parameters
  integer, parameter :: n_values = 10
  real(kind=default), parameter :: sqrts_min = 350, sqrts_max = 800

  ! Variables
  integer :: i
  character(len=14+3) :: filename

  ! Tables
  real(kind=default), dimension(n_values) :: &
      sqrts, integral, error, chi2, efficiency

  ! Main loop
  do i = 1, n_values

     ! Calculate energy, set filename accordingly
     sqrts(i) = ((i-1) * sqrts_max + (n_values-i) * sqrts_min) / (n_values-1)
     write(filename, "(A14,I3)")  "whizard_sqrts=", nint(sqrts(i))
     call whizard_set_filename (filename)

     ! Read input and override energy setting
     write (*,*)
     write (*, "(A,1x,I3,1x,A)") "*** sqrts =", nint(sqrts(i)), "***"
     call whizard_read_input
     whizard_input%sqrts = sqrts(i)

     ! Integrate, record result, generate events
     call whizard_integrate
     call whizard_get_results (i, integral(i), error(i), chi2(i), efficiency(i))
```

[32] To access the module information, several subdirectories must be included in the module search path of the compiler. Inspect the setting of F90INCL in the Makefile in the `whizard-src` subdirectory.

[33] This works for a Fortran 95 main program; the analogous steps for a main program in a different language will require modifications of `whizard.ld`, in particular the additional linking of the compiler-specific Fortran 95 runtime library.

```
      call whizard_generate
      call whizard_end

  end do

  ! Write table
  write (*,*)
  write (*, "(A)") "*** Results:"
  write (*, "(1x,A15,2x,A25,2x,A25)") "sqrts [GeV]", &
       "Total cross section [fb]", "Integration error [fb]"
  do i = 1, n_values
     write (*, "(1x,G15.5,2x,G25.5,2x,G25.5)") &
       sqrts(i), integral(i), error(i)
  end do
  write (*, "(A)") "*** End of WHIZARD runs"

end program my_main
```

The meaning of the individual subroutine calls is described in detail in the following section. Note that parameters in the input block can be accessed via the object `whizard_input` which is a module variable of the `whizard` module. More exactly, variables in the `process_input`, `integration_input`, `simulation_input` and `diagnostics_input` blocks can be accessed as above

```
whizard_input%sqrts = sqrts
```

while variables in the `parameter_input` block are accessed like

```
whizard_input%par%mh = 115are
```

and beam parameters are accessed like

```
whizard_input%beam(1)%PDF_nset = 43
```

5.9 Interface of the WHIZARD module

All subroutines described in this section are accessed via the module `whizard`, which is included in the library `whizard.a`. Therefore, a main program written in Fortran 95 has to contain the line

```
use whizard
```

in the preamble.[34] For other programming languages, the subroutines are accessed by a platform-dependent prefix. For instance, the NAG compiler preprends the string `whizard_MP_` to all names in the module `whizard`, so the subroutine `whizard_integrate` is accessed by the name `whizard_MP_whizard_integrate`. The access to arguments in subroutine calls may also be platform-dependent.

[34] See footnote 32.

5.9.1 Auxiliary routines

The following routines are needed only if some non-standard behavior of `WHIZARD` is intended. They closely match the command-line arguments described in Sec. 5.6.

`whizard_set_directory` : Set a working directory different from the default one.

Invocation: `call whizard_set_directory (`*directory*`)`

Arguments:

directory : string (input parameter)

`whizard_set_filename` : Set a specific filename to be used instead of `whizard` for all input and output files during this run. This is one way of avoiding files getting overwritten. If an input file with specific filename does not exist, the generic input files `whizard.in` etc. will be used. The output files will always get the specific name, if any.

Invocation: `call whizard_set_filename (`*filename*`)`

Arguments:

filename : string (input parameter)

`whizard_set_input_file` : Define the name of the main input file. Needed only if it is different from the default. The extension `.in` will be appended.

Invocation: `call whizard_set_input_file (`*filename*`)`

Arguments:

filename : string (input parameter)

`whizard_enable_input_file` : Enable/disable reading of the main input file. If the logical argument is `.false.`, calls to `whizard_read_input` will not read any file. This makes sense if parameters are to be set by different means, in particular by calls to `whizard_set_input`.

Invocation: `call whizard_enable_input_file (`*flag*`)`

Arguments:

flag : logical true or false (input parameter)

`whizard_set_input` : Set input parameters directly. Each argument is a string of *parameter = value* definitions in namelist format, as it would appear in the input file enclosed by "`&xxx`" and "`/`", where `xxx` is the respective input block (e.g., `process_input`). The strings can be empty.

Invocation: `call whizard_set_input (`*process_input*`,` *integration_input*`,`
simulation_input`,` *diagnostics_input*`,` *parameter_input*`,`
beam_input1`,` *beam_input2*`)`

Arguments:

process_input : namelist string (input parameter)

integration_input : namelist string (input parameter)

simulation_input : namelist string (input parameter)

diagnostics_input : namelist string (input parameter)

parameter_input : namelist string (input parameter)

beam_input1 : namelist string (input parameter)

beam_input2 : namelist string (input parameter)

5.9.2 Standard routines

The following routines are called, directly or indirectly, in standard `WHIZARD` runs. The explicit interface allows for calling them directly without reference to the main program.

`whizard_read_input` : Start a new instance of WHIZARD and terminate any pending run. Read the input file. If no specific filename has been given, this is called `whizard.in`. If this routine is not called explicitly, it will be called automatically by `whizard_integrate`. Calling it explicitly allows for overriding input data before they are fed into the integration pass.

Invocation: `call whizard_read_input`

Arguments: none.

`whizard_number_processes` : Return the number of processes specified in the input file.

Invocation: `call whizard_number_processes (`*number*`)`

Arguments:

number : integer (output parameter)

`whizard_integrate` : Calculate the total cross section resp. read pre-generated results from file, as required by the input data. This routine reads the model definition file `whizard.mdl`, the cut configuration file `whizard.cut1` and, if required, it reads/writes the grids `whizard.grc` and `whizard.grb`. If this routine is not called explicitly, it will be called automatically by the event generation routines.

Invocation: `call whizard_integrate`

Arguments: none.

`whizard_get_results` : Read out the current integration results for the indicated process: Total cross section, error estimate, χ^2 value, reweighting efficiency estimate. If *process_index* is zero, return the total integral, error and efficiency average for all activated processes. The χ^2 value is zero in that case. If no integration has yet been performed, all returned numbers are zero. If the *process_index* is larger than the number of processes (as returned by `whizard_number_processes`), an error is issued.

The returned numbers are of the default real kind, which is double precision under normal circumstances. The default real kind may be accessed in a Fortran 95 program by a `use kinds` directive and including the `kinds-src` subdirectory in the module search path.

Invocation: `call whizard_get_results (`*process_index, integral, error, chi2, efficiency*`)`

Arguments:

process_index : integer (input variable)

integral : default real (output variable)

error : default real (output variable)

chi2 : default real (output variable)

efficiency : default real (output variable)

`whizard_generate` : Generate a fixed number of events resp. read them from file, as specified by the input data. This routine reads the cut/analysis configuration `whizard.cut5` and reads/writes the event file(s) `whizard.evt` and `whizard.evx`, if required. Furthermore, it can write histogram data in `whizard-plots.dat`. This subroutine is equivalent to consecutive calls of `whizard_events_begin`, `whizard_event` (multiple times), and `whizard_events_end` (see below).

Invocation: `call whizard_generate`

Arguments: none

`whizard_events_begin` : Prepare event generation for a `n_calls` events. It is assumed that the main program contains a loop with exactly `n_calls` to `whizard_event`. However, in case of unweighted events generation, this number is used only for opening a STDHEP format file, so the value is irrelevant if that option is not used.

Invocation: `call whizard_events_begin (`*n_calls*`)`

Arguments:

n_calls : 64-bit integer (input parameter)

`whizard_event` : Generate a single (unweighted) event. The event is stored in the standard `HEPEVT` COMMON block. If requested by the input data, it is also written to file. To make use of the current event, the caller routine must be able to access `HEPEVT`.

Invocation: `call whizard_event`

Arguments: none

`whizard_events_end` : Finalize event generation.

Invocation: `call whizard_events_end`

Arguments: none

`whizard_end` : Finalize the `WHIZARD` run. Close all open files and deallocate memory.

Invocation: `call whizard_end`

Arguments: none

6 Examples

In the following sections five realistic applications of WHIZARD are presented: first, three LEP examples, namely, Higgs production in the four-fermion channels, W production (four fermions), and strong WW scattering (six fermions). With certain refinements and modifications, they could be used for actual physical applications. However, here they are intended as a form of tutorial, showing typical setups and common problems. We then switch to two LHC examples, the Jacobian peak or endpoint in W production and Drell-Yan production of a heavy Z' vector boson. Note that WHIZARD 1 was intended for lepton collider simulations basically, a lot of hadron collider physics does work perfectly with this version, but WHIZARD 2 has been specifically designed for hadron collider physics. So we strongly recommend using WHIZARD 2 in such cases. We nevertheless present the two LHC examples here.

6.1 Higgs production at LEP

Let us simulate the production of a 115 GeV Standard Model Higgs in e^+e^- collisions at 209 GeV. With this mass, the Higgs boson decays mainly into $\bar{b}b$ with a small fraction of $\tau^+\tau^-$ decays. Here, we only consider the signal together with its irreducible background, contributing to the processes

$$\begin{aligned} e^-e^+ &\to \nu\bar{\nu}b\bar{b} \\ &\to q\bar{q}b\bar{b} \\ &\to b\bar{b}b\bar{b} \\ &\to \mu^-\mu^+b\bar{b} \\ &\to e^-e^+b\bar{b} \\ &\to q\bar{q}\tau^-\tau^+ \end{aligned}$$

where $q = u, d, s, c$ and $\nu = \nu_e, \nu_\mu, \nu_\tau$. This list of processes directly translates into the configuration file whizard.prc:

```
# WHIZARD configuration file

# The selected model
model SM

alias q u:d:s:c
alias Q U:D:S:C
alias n n1:n2:n3
alias N N1:N2:N3

# Processes
#  Methods: omega=O'Mega)
```

```
# Tag     In      Out      Method Option
#=================================================
# On-shell process:
zh              e1,E1   Z,H             omega
# Full four-fermion matrix elements:
nnbb            e1,E1   n,N,b,B         omega
qqbb            e1,E1   q,Q,b,B         omega   c
bbbb            e1,E1   b,B,b,B         omega   c
eebb            e1,E1   e1,E1,b,B       omega
mmbb            e1,E1   e2,E2,b,B       omega
qqtt            e1,E1   q,Q,e3,E3       omega
bbtt            e1,E1   b,B,e3,E3       omega
```

Here, we use flavor summation for the quark and missing-energy channels. After this file has been saved in the `conf` subdirectory, we should run `configure` in the directory where we had unpacked the distribution, if not already done,

```
> ./configure
```

and make and install the executable

```
> make install
```

After this step is completed, we are ready for running the program.

6.1.1 The on-shell process

All necessary files have now been installed in the `results` subdirectory:

```
> cd results
> ls
Makefile      whizard       whizard.cut5  whizard.mdl
Makefile.in   whizard.cut1  whizard.in    whizard.prc
```

The file `whizard.prc` is a copy of the one we prepared above. `whizard.mdl` contains all SM-specific definitions. Here, only the vertex list contained within is used for phase space generation; usually we don't need to modify it. The two cut configuration files are empty. The file which is of interest for us now is `whizard.in` which we have to edit:

```
&process_input
process_id = "zh"
sqrts = 209
/

&integration_input
stratified = F
```

```
/

&simulation_input /

&diagnostics_input /

&parameter_input
MH = 115
wH = 0.3228E-02
Mb = 2.9
Me = 0
Ms = 0
Mc = 0
/

&beam_input /

&beam_input /
```

As a warm-up, we examine the on-shell process $e^-e^+ \to ZH$, labeled `zh`. We insert the c.m. energy 209 GeV and the Higgs mass 115 GeV. The next two parameters are not relevant here, but will be needed for the other processes: The Higgs width value `wH` is the one returned by HDECAY [8]. The b mass `Mb` is the running mass evaluated at a scale around m_H. This is important since the Hbb coupling is proportional to m_b.[35] The setting `stratified=F` calls for importance sampling instead of stratified sampling (see below). All other fields are left empty.

Having saved `whizard.in`, the program can be started:

```
> ./whizard
```

(if you like, you can also type `make run`). Since this process is trivial, the results are there immediately:

```
! WHIZARD 1.97 (May 31 2011)
! Reading process data from file whizard.in
! Wrote whizard.out
! Reading phase space configurations from file whizard.phx
!
! Process nnbb:
!    e a-e -> nu_e a-nu_e   b a-b
!    e a-e -> nu_mu a-nu_mu   b a-b
!    e a-e -> nu_tau a-nu_tau   b a-b
!   32  16 ->      1        2   4   8
! Process energy set to     209.00     GeV
```

[35] For a genuine analysis, one should inspect the underlying assumptions and try to find a consistent parameter set. Since the `WHIZARD` matrix elements are evaluated at tree level, some higher-order corrections should not be included in the Higgs width.

```
!    12 phase space channels found for process nnbb
! Scanning phase space channels for equivalences ...
! Phase space:      12 equivalence relations found.
!
! Created grids:     12 channels,  8 dimensions with 20 bins
!
! WHIZARD run for process nnbb:
!=============================================================================
! It      Calls  Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!-----------------------------------------------------------------------------
! Reading cut configuration data from file whizard.cut1
! No cut data found for process nnbb
! Preparing (fixed weights):   1 sample of      20000 calls ...
   1      20000  1.0948826E+02  1.94E+00    1.77    2.51*  3.93    0.00   1
!-----------------------------------------------------------------------------
! Adapting (variable wgts.):  10 samples of      20000 calls ...
   2      20000  1.0834009E+02  1.92E+00    1.77    2.50*  3.88
   3      20000  1.0638224E+02  7.58E-01    0.71    1.01* 14.69
   4      20000  1.0667771E+02  6.08E-01    0.57    0.81* 19.32
   5      20000  1.0672584E+02  5.62E-01    0.53    0.75* 19.40
   6      20000  1.0562583E+02  5.39E-01    0.51    0.72* 19.10
   7      20000  1.0656915E+02  5.45E-01    0.51    0.72  15.70
   8      20000  1.0690159E+02  5.39E-01    0.50    0.71* 18.15
   9      20000  1.0706322E+02  5.39E-01    0.50    0.71* 18.44
  10      20000  1.0656723E+02  5.29E-01    0.50    0.70* 18.90
  11      20000  1.0630854E+02  5.23E-01    0.49    0.70* 20.26
!-----------------------------------------------------------------------------
! Integrating (fixed wgts.):   3 samples of      20000 calls ...
  12      60000  1.0655809E+02  3.04E-01    0.28    0.70  15.36    1.57   3
!-----------------------------------------------------------------------------
!
! Time estimate for generating 10000 unweighted events:    0h 00m 12s
!=============================================================================
! Summary (all processes):
!-----------------------------------------------------------------------------
! Process ID     Integral[fb]  Error[fb]   Err[%]        Frac[%]
!-----------------------------------------------------------------------------
  nnbb           1.0655809E+02  3.04E-01    0.28         100.00
!-----------------------------------------------------------------------------
  sum            1.0655809E+02  3.04E-01    0.28         100.00
!=============================================================================
! Wrote whizard.out
! Integration complete.
! No event generation requested
```

The cross section is[36]

$$\sigma = 106.56 \pm 0.30 \text{ fb}$$

This calculation takes roughly a minute on a PC (Intel Core2 2.5 GHz, gfortran compiler). Of course, if we invest more CPU time, we can improve the accuracy, as explained in the previous

[36]This number should be taken with a grain of salt. See footnote 35.

subsection.

In inspecting the output, the column tagged `Err/Exp` is of main interest (read: relative error over expected relative error). This is the estimated relative error multiplied by the square root of the number of calls, a number which should be of the order one. Each time this error improves, the entry is marked with a star. The best grid so far (the last one with a star) will be used for integration, and later for event generation. As evident from the column `Eff[%]`, the estimated reweighting efficiency improves as well. In the last column the χ^2 value divided by the number of iterations is shown. This value is meaningful only if there is more than one iteration in the integration step. This is not the case here, so it is zero in our example.

Let us now generate events. We define a nonzero luminosity (100 fb^{-1}, which was more than 100 times the LEP integrated luminosity per year):

```
&process_input
process_id = "nnbb"
sqrts = 209
luminosity = 100
/
```

We do not want to repeat the adaptation and integration steps, therefore we read in the previously adapted grids

```
&integration_input
read_grids = T
/
```

Now running `WHIZARD` will result in a repetition of the previous result, before event generation is started:

```
! Reading analysis configuration data from file whizard.cut5
! No analysis data found for process nnbb
! Event sample corresponds to luminosity [fb-1] =   100.0
! Event sample corresponds to       69367  weighted events
! Generating      10656 unweighted events ...
! Event generation finished.
!=============================================================================
! Analysis results for process nnbb:
! It      Events Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!-----------------------------------------------------------------------------
  13      10656  1.0655809E+02  1.03E+00    0.97    1.00 100.00
!-----------------------------------------------------------------------------
! Warning: Excess events:    7.9       (    0.07% )  | Maximal weight:  1.25
```

The generation of 10,656 events takes one more minute in this case. No further analysis has been requested, so we just get the total cross section again. By definition, this is equal to the previously calculated cross section, but the error now corresponds to the actual event sample.

There are some excess events (weight greater than one), but the effect of this excess being dropped is considerably less than the statistical error of the event sample. If necessary, the

excess could be removed by setting `safety_factor` to a value greater than one (e.g., 1.1), at the expense of extra CPU time since the reweighting efficiency is reduced by this factor.

The event sample can be further analyzed: Let us plot the missing invariant mass distribution and the dijet invariant mass distribution, which should exhibit peaks at the Z and Higgs mass. To this end, we tell `WHIZARD` to reread the generated event sample

```
&simulation_input
read_events = T
/
```

and make up an analysis configuration file `whizard.cut5`:

```
# cut/histogram configuration file
# e- e+ -> nu nubar b bbar
# 32 16    1    2   4  8
process nnbb, qqbb, bbbb, eebb, mmbb, qqtt, bbtt
  histogram M of 3 within 0 209 nbin 40
and
  histogram M of 12 within 0 209 nbin 40
and
  cut M of 12 within 114 116
  histogram M of 3 within 0 209 nbin 40
```

This type of analysis can serve for all processes, therefore we have added the other tags. The last histogram counts only those events which have a $b\bar{b}$ invariant mass close to the Higgs mass. The numbers result from adding up the binary codes of the external particles, which are shown as a the comment in the file header.

Running `WHIZARD` this time takes almost no time since only the previously generated files have to be read:

```
 ! Reading analysis configuration data from file whizard.cut5
 ! Found   3 analysis configuration datasets.
 ! Looking for raw event file whizard.evx ...
 ! Event sample corresponds to luminosity [fb-1] =   100.0
 ! Event sample corresponds to       69367  weighted events
 ! Reading      10656 unweighted events ...
 ! Event generation finished.
 !=============================================================================
 ! Analysis results for process nnbb:
 ! It      Events Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
 !-----------------------------------------------------------------------------
   13      10656  1.0655809E+02  1.03E+00    0.97    1.00 100.00
 !-----------------------------------------------------------------------------
 ! Warning: Excess events:    7.9       (    0.07% )  | Maximal weight:  1.25
 !=============================================================================
 ! Analysis results for process nnbb:
 ! It      Events Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
 !-----------------------------------------------------------------------------
```

```
  13      10656  1.0655809E+02  1.03E+00    0.97    1.00 100.00
!---------------------------------------------------------------------------
! Warning: Excess events:    7.9       (    0.07% )  | Maximal weight:  1.25
!===========================================================================
! Analysis results for process nnbb:
!
! Additional cuts:
! integration pass 5
cut M of   12      within  1.14000E+02  1.16000E+02
! It      Events Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!---------------------------------------------------------------------------
  13       2166  2.1659612E+01  4.65E-01    2.15    1.00  20.33
!---------------------------------------------------------------------------
! Warning: Excess events:    1.5       (    0.07% )  | Maximal weight:  1.05
```

From the last line we read off that the actual contribution of Higgs production in this channel is 2166 events (or 20.33 %). Here the efficiency is not the reweighting efficiency as before, but the fraction of events remaining after cuts. The histograms can now be found in the output file `whizard.nnbb.dat`:

```
!===========================================================================
! WHIZARD 1.97 (May 31 2011)
! Process nnbb:
!    e a-e -> nu_e a-nu_e   b a-b
!    e a-e -> nu_mu a-nu_mu   b a-b
!    e a-e -> nu_tau a-nu_tau   b a-b
!   32  16 ->      1        2   4   8
! Analysis results for process nnbb:
!
! Histograms:
!
histogram M of   3     within  0.00000E+00  2.09000E+02  nbin   40
  2.61250000        0.00000000        0.00000000        0.00000000
  7.83750000        1.00000000        1.00000000        0.00000000
  13.0625000        2.00000000        1.41421356        0.00000000
  18.2875000        3.00000000        1.73205081       0.220852468
  23.5125000        4.00000000        2.00000000        0.00000000
  28.7375000        7.00000000        2.64575131        0.00000000
  33.9625000        10.0000000        3.16227766       0.379512096E-01
  39.1875000        13.0000000        3.60555128        0.00000000
  44.4125000        20.0000000        4.47213595       0.455956036E-02
  49.6375000        28.0000000        5.29150262       0.180557973
  54.8625000        36.0000000        6.00000000       0.452075346E-03
  60.0875000        49.0000000        7.00000000        0.00000000
  65.3125000        63.0000000        7.93725393       0.131335186
  70.5375000        72.0000000        8.48528137       0.922046560
  75.7625000        154.000000        12.4096736        0.00000000
  80.9875000        274.000000        16.5529454        0.00000000
  86.2125000        1071.00000        32.7261363        1.47417846
.....
```

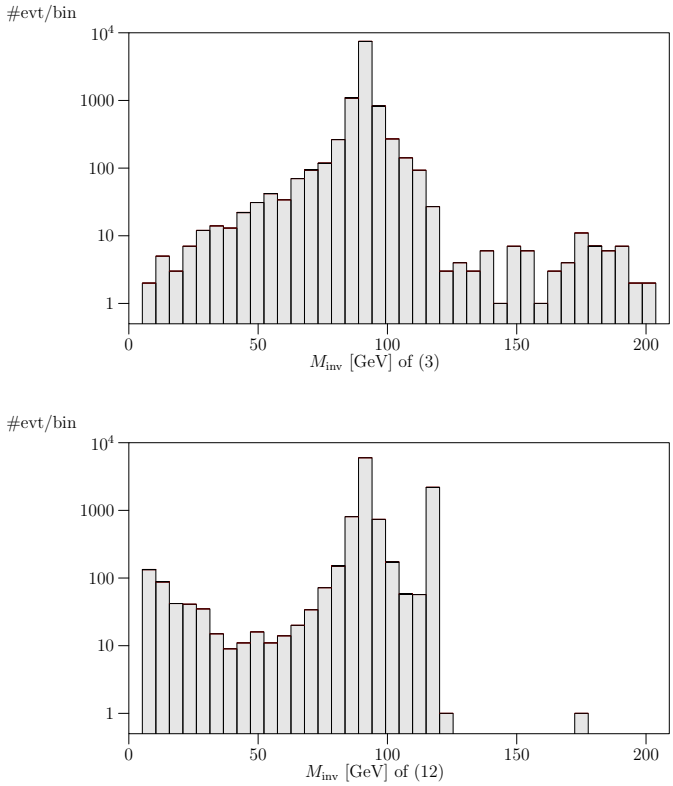


Figure 16: *Histograms for the process* $e^-e^+ \to \nu\bar{\nu}b\bar{b}$ *at* $\sqrt{s} = 209$ *GeV with* $m_H = 115$ *GeV. Top: Dijet invariant mass distribution. Bottom: Missing invariant mass distribution.*

In these histograms, the first column is the bin midpoint. The second column is the number of entries, the third column the statistical error on this number ($\sqrt{N}$). The fourth column shows the excess events, which introduce an additional error. However, for each bin this contribution is negligible compared to the statistical error which is present anyway.

The actual histograms are shown in Fig. 16. (For better illustration, we have changed the linear scale to a logarithmic one; this is done in the file `whizard-plots.tex` by the manual replacement of the lines [37]

```
setup(linear,linear); graphrange (#0.00,#0), (#209.,??);
```

by

```
setup(linear,log); graphrange (#0.00,#.5), (#209.,#1e4);
```

and a subsequent rerun)

```
> make plots
```

In the dijet invariant mass distribution the Z and Higgs peaks are clearly visible. At low dijet invariant mass there is a tail of continuum $b\bar{b}$ production. In the missing mass distribution the peak is at the Z mass.

[37] Note that in `WHIZARD` 2 this can be directly demanded by flags in the input file.

6.1.2 The four-jet channel

There are two four-jet channels

$$e^-e^+ \to q\bar{q}b\bar{b}, \quad b\bar{b}b\bar{b}.$$

If the light quark masses are set to zero (see the input file above), all light quark channels can be treated in a single run. For unequal masses, flavor summation is not possible, therefore the b channel has to be treated separately. (We can't set the b mass to zero since this would remove the b Yukawa coupling.)

Running the program as before for the process `qqbb` with the given input and analysis configuration files takes about a few minutes in total. The cross section result is

$$\sigma = 284.05 \pm 0.71 \text{ fb}$$

and 28406 events are generated in this run, 4637 of them within the Higgs mass window.

To obtain a finite result, `WHIZARD` had to insert a cut of 10 GeV on the dijet invariant mass of the light quarks. This cut is written into the file `whizard.qqbb.cut0`:

```
! Automatically generated set of cuts
! Process qqbb:
!    e a-e ->   u a-u   b a-b
!    e a-e ->   d a-d   b a-b
!    e a-e ->   s a-s   b a-b
!    e a-e ->   c a-c   b a-b
!   32  16 ->   1   2   4   8
process qqbb
cut M of    3      within  1.00000E+01  1.00000E+99
```

The cut could be modified either by changing the default value in `whizard.in`, e.g.

```
&integration_input
...
default_jet_cut = 5
/
```

or by providing a non-empty file `whizard.cut1` of the same format, with a different set of cuts. If an appropriate process entry is found in this file, any default cuts are ignored.

The $b\bar{b}b\bar{b}$ channel is similar, although the presence of identical particles in the final state makes phase space more complicated. The adaptation again works well and we get

$$\sigma = 44.71 \pm 0.12 \text{ fb} \tag{10}$$

6.1.3 The lepton channels

In the $\mu^+\mu^-$ channel adaptation takes somewhat longer due to the small muon mass, if no cuts are provided. However, the 10 iterations of the adaptation step are sufficient, and in the final integration step, the result

$$\sigma = 52.83 \pm 0.16 \text{ fb}$$

is perfectly stable. With suitable cuts, we can set the muon mass to zero, which as a side effect speeds up the calculation by a factor of two.

A more difficult task is to get to a stable result for the cross section of the process

$$e^- e^+ \to e^- e^+ \bar{b} b$$

without cuts, but with nonzero electron mass. The default choice for the number of calls and iterations does not suffice here:

```
! WHIZARD run for process eebb:
!===============================================================================
! It      Calls  Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!-------------------------------------------------------------------------------
! Reading cut configuration data from file whizard.cut1
! Replacing default cuts by user-defined cuts.
! Preparing (fixed weights):   1 sample of      20000 calls ...
   1      20000  1.0106043E+03  3.58E+02   35.42   50.10*  0.22    0.00   1
!-------------------------------------------------------------------------------
! Adapting (variable wgts.):  30 samples of      20000 calls ...
   2      20000  1.1608216E+03  5.28E+02   45.46   64.29   0.14
   3      20000  9.4934352E+02  2.17E+02   22.83   32.28*  0.09
   4      20000  1.6287972E+03  2.77E+02   17.00   24.04*  0.12
   5      20000  2.3970665E+03  6.33E+02   26.41   37.35   0.07
   6      20000  3.8934769E+03  1.27E+03   32.57   46.06   0.03
   7      20000  1.2274732E+04  9.29E+03   75.71  107.07   0.05
   8      20000  5.5859667E+03  2.25E+03   40.21   56.87   0.06
   9      20000  3.8208742E+03  5.29E+02   13.84   19.57*  0.13
  10      20000  4.5776710E+03  6.04E+02   13.20   18.67*  0.27
  11      20000  1.2110487E+04  5.47E+03   45.16   63.86   0.04
.....
```

Apparently, convergence is not reached. One possibility to proceed is to reuse these grids, but do further iterations:

```
&integration_input
calls = 1 20000 30 20000 3 20000
read_grids = T
/
```

For the first iterations the old results are copied. The additional iterations now find convergence, although the final accuracy and efficiency are not really satisfactory:

```
.....
  28      20000  7.5246100E+03  1.72E+02    2.29    3.24   1.17
  29      20000  7.7685827E+03  2.05E+02    2.64    3.73   0.93
  30      20000  8.0304810E+03  3.52E+02    4.38    6.19   0.60
  31      20000  7.7031026E+03  2.21E+02    2.87    4.06   0.81
!-------------------------------------------------------------------------------
! Integrating (fixed wgts.):   3 samples of      20000 calls ...
  32      60000  7.2436310E+03  1.39E+02    1.91    4.69   0.31    2.37   3
!-------------------------------------------------------------------------------
```

To get a more precise result, we should also increase the number of calls per iteration. This costs additional CPU time.

Fortunately, there is a shortcut. The problem of this calculation is the fact that a large part of the cross section comes from the region of very forward electrons. Furthermore, there is a piece which corresponds to $b\bar{b}\gamma^*$ production where the virtual photon splits into the electron-positron pair. These small regions are sampled only after several iterations and only then have sufficiently adapted grids. The mappings built into `WHIZARD` assume the parameters `default_Q_cut` and `default_mass_cut` as typical scales for these subprocesses. The implicit assumption, which works in many cases, is that this is not too far from the actual cuts. However, if there are no cuts, the assumption is inadequate. We rather should set those parameters to zero

```
&integration_input
  default_Q_cut = 0
  default_mass_cut = 0
/
```

(which actually replaces them by some finite small value), and start again:

```
! WHIZARD run for process eebb:
!=============================================================================
! It      Calls  Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!-----------------------------------------------------------------------------
! Reading cut configuration data from file whizard.cut1
! Replacing default cuts by user-defined cuts.
! Preparing (fixed weights):   1 sample of      20000 calls ...
   1      20000  8.6041111E+03  4.17E+03   48.43   68.48*  0.16    0.00   1
!-----------------------------------------------------------------------------
! Adapting (variable wgts.):  10 samples of      20000 calls ...
   2      20000  1.0198520E+04  3.59E+03   35.25   49.84*  0.19
   3      20000  7.1720651E+03  4.57E+02    6.37    9.00*  0.44
   4      20000  8.0690070E+03  3.42E+02    4.24    6.00*  0.39
   5      20000  7.5135901E+03  3.94E+02    5.24    7.41   0.40
   6      20000  8.0065259E+03  4.47E+02    5.58    7.89   0.31
   7      20000  7.4418114E+03  2.71E+02    3.65    5.16*  0.88
   8      20000  7.8493882E+03  1.78E+02    2.27    3.20*  1.26
   9      20000  7.3683498E+03  1.27E+02    1.72    2.43*  2.27
  10      20000  7.4932575E+03  1.13E+02    1.51    2.14*  2.55
  11      20000  7.7184631E+03  1.45E+02    1.88    2.66   2.26
!-----------------------------------------------------------------------------
! Integrating (fixed wgts.):   3 samples of      20000 calls ...
  12      60000  7.5716096E+03  1.13E+02    1.49    3.66   0.09    1.59   3
!-----------------------------------------------------------------------------
```

This time, the difficult regions are sampled from the beginning.

With standard cuts on the electron energies and angles, this process is well-behaved. The extra effort is needed only if we are particularly interested in the events with electron and positron going in the very forward and backward regions, respectively.

6.2 6-fermion production: Higgs pairs

A process which is of interest at a future Linear Collider is Higgs pair production, which is sensitive to the Higgs trilinear coupling. A process definition file `whizard.prc` for this process could look like

```
# WHIZARD configuration file

alias q u:d:s:c
alias Q U:D:S:C

model   SM
#============================================================
# On-shell process:
zhh             e1,E1   Z,H,H         omega

# Full six-fermion matrix elements:
qqbbbb          e1,E1   q,Q,b,B,b,B   omega       c
```

For simplicity, we consider only six-quark production where four quarks are b quarks (the main decay channel of the Higgs pair), and the remaining quark pair is light.

The input file `whizard.in` is prepared for the six-fermion process at $\sqrt{s} = 500$ GeV with default settings. We would like to generate an event sample corresponding to 10 ab^{-1}.

```
&process_input
process_id = "qqbbbb"
sqrts = 500
luminosity = 10000
/

&integration_input /

&simulation_input /

&diagnostics_input /

&parameter_input
mH = 115
wH = 0.3228E-02
mb = 2.9
me = 0
ms = 0
mc = 0
/
```

```
&beam_input /
```

```
&beam_input /
```

Without further considerations, we switch into the `results` subdirectory and start integration and event generation

```
> ./whizard
```

After the initial message

```
! WHIZARD 1.97 (May 31 2011)
! Reading process data from file whizard.in
! Wrote whizard.out
!
! Process qqbbbb:
!    e a-e ->   u a-u   b a-b   b a-b
!    e a-e ->   d a-d   b a-b   b a-b
!    e a-e ->   s a-s   b a-b   b a-b
!    e a-e ->   c a-c   b a-b   b a-b
!  128  64 ->   1   2   4   8  16  32
! Process energy set to     500.00     GeV
! Reading vertices from file whizard.mdl ...
! Model file:       54 trilinear vertices found.
! Model file:       54 vertices usable for phase space setup.
! Generating phase space channels for process qqbbbb...
```

the program needs quite some time for generating the phase space configuration (this would be faster without flavor summation). Fortunately, the configuration is written to the file `whizard.phx` and `WHIZARD` will reuse it when possible.

After this step is finished, a default cut for the light quark pair is inserted and integration is started.

```
! Phase space:     342 phase space channels generated.
! Scanning phase space channels for equivalences ...
! Phase space:    1368 equivalence relations found.
! Note: This cross section may be infinite without cuts.
! Wrote default cut configuration file whizard.qqbbbb.cut0
! Note: This cross section may be infinite without cuts.
! Wrote default cut configuration file whizard.qqbbbb.cut0
! Note: This cross section may be infinite without cuts.
! Wrote default cut configuration file whizard.qqbbbb.cut0
! Note: This cross section may be infinite without cuts.
! Wrote default cut configuration file whizard.qqbbbb.cut0
! Wrote phase space configurations to file whizard.phx
!
! Created grids:    342 channels, 14 dimensions with 20 bins
!
! WHIZARD run for process qqbbbb:
!===============================================================================
! It      Calls  Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
```

```
!------------------------------------------------------------------------
! Reading cut configuration data from file whizard.cut1
! No cut data found for process qqbbbb
! Using default cuts.
cut M of    3      within  1.00000E+01  1.00000E+99
! Preparing (fixed weights):   1 sample of     100000 calls ...
```

The whole adaptation and integration run takes a considerable amount of CPU time (about one day on an Xeon or Alpha processor). If only one quark flavor were considered, this could be reduced by a factor of four (currently, `WHIZARD/O'Mega` does not take advantage of the fact that some matrix elements with different flavor content are in fact identical. Such a mechanism, known as cross-flavor common subexpression elimination has been incorporated into `WHIZARD` 2.).

A usable result is already reached after about 10 iterations, with considerable fluctuation around the optimal grid[38]:

```
   1     100000  1.4904184E+00  1.97E-01   13.21   41.77*  0.54    0.00   1
!------------------------------------------------------------------------
! Adapting (variable wgts.):  20 samples of     100000 calls ...
   2     100000  1.5389248E+00  3.52E-01   22.86   72.29   0.52
   3     100000  1.5009010E+00  7.44E-02    4.96   15.67*  0.60
   4     100000  1.6253297E+00  7.43E-02    4.57   14.45*  0.68
   5     100000  1.6136791E+00  4.64E-02    2.87    9.08*  0.70
   6     100000  2.2312050E+00  6.57E-01   29.46   93.15   0.51
   7     100000  1.5917687E+00  3.05E-02    1.92    6.07*  1.09
   8     100000  1.6079748E+00  4.15E-02    2.58    8.16   0.91
   9     100000  1.5235805E+00  3.38E-02    2.22    7.03   1.21
  10     100000  1.6966923E+00  7.18E-02    4.23   13.38   1.16
  11     100000  1.5519629E+00  2.22E-02    1.43    4.52*  1.68
  12     100000  1.5970681E+00  2.47E-02    1.55    4.90   1.74
  13     100000  1.6847993E+00  5.81E-02    3.45   10.90   1.67
  14     100000  1.5552737E+00  1.64E-02    1.05    3.33*  2.55
  15     100000  1.6117607E+00  1.84E-02    1.14    3.61   2.67
  16     100000  1.6167684E+00  2.17E-02    1.34    4.24   2.63
  17     100000  1.6052581E+00  1.94E-02    1.21    3.83   2.82
  18     100000  1.6101957E+00  1.95E-02    1.21    3.84   3.10
  19     100000  1.6045048E+00  1.54E-02    0.96    3.03*  3.20
  20     100000  1.5782323E+00  1.44E-02    0.91    2.88*  3.78
  21     100000  1.5855381E+00  1.33E-02    0.84    2.65*  3.76
!------------------------------------------------------------------------
! Integrating (fixed wgts.):   3 samples of     100000 calls ...
```

Nevertheless, the best grid obtained so far can safely be used for event generation. The final estimate for the integral is

```
  22     300000  1.5716132E+00  6.93E-03    0.44    2.41*  2.80    0.46   3
!------------------------------------------------------------------------
!
! Time estimate for generating 10000 unweighted events:    1h 19m 27s
```

[38] By reducing the parameter `weights_power` in the input file, these fluctuations can be somewhat dampened.

However, we don't need that many events: simulating 10 ab^{-1} now takes only half an hour:

```
! Event sample corresponds to luminosity [fb-1] =  0.1000E+05
! Event sample corresponds to      561275  weighted events
! Generating      15716 unweighted events ...
! Event generation finished.
!============================================================================
! Analysis results for process qqbbbb_full:
! It      Events Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!----------------------------------------------------------------------------
  23      15716  1.5716132E+00  1.25E-02    0.80    1.00 100.00
!----------------------------------------------------------------------------
! Warning: Excess events:  273.0       (    1.74% )  | Maximal weight: 45.62
```

If we wish to know how many of those events are originating from HH pairs, we should set up an analysis configuration file `whizard.cut5` like this

```
! e- e+ -> q qbar b bbar b bbar
!128 64    1  2   4  8  16  32
process qqbbbb
  cut M of 12 within 114 116
  cut M of 48 within 114 116
and
  cut M of 36 within 114 116
  cut M of 24 within 114 116
```

and rerun the program, setting `read_grids=T` and `read_events=T`. The result is

```
 ! Analysis results for process qqbbbb_full:
 !
 ! Additional cuts:
 ! integration pass 5
 cut M of   12      within  1.14000E+02  1.16000E+02
 cut M of   48      within  1.14000E+02  1.16000E+02
 ! It      Events Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
 !----------------------------------------------------------------------------
   23        195  1.9500164E-02  1.40E-03    7.16    1.00   1.24
 !----------------------------------------------------------------------------
 ! Warning: Excess events:    1.4       (    0.70% )  | Maximal weight:  1.27
 !============================================================================
 ! Analysis results for process qqbbbb_full:
 !
 ! Additional cuts:
 ! integration pass 5
 cut M of   36      within  1.14000E+02  1.16000E+02
 cut M of   24      within  1.14000E+02  1.16000E+02
 ! It      Events Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
 !----------------------------------------------------------------------------
   23        167  1.6700140E-02  1.29E-03    7.74    1.00   1.06
 !----------------------------------------------------------------------------
 ! Warning: Excess events:    0.0       (    0.00% )  | Maximal weight:  1.00
```

The two event samples may be added (assuming that no events pass both cuts simultaneously), to yield 362 "signal" events.

The stability of the result and the computing time can be improved by reducing the number of phase space channels (see Sec. 4.6.2).

6.3 Vector boson scattering: polarization and beamstrahlung

In case no light Higgs boson exists, one will try to measure vector boson scattering at high-energy colliders, e.g.

$$W^+W^- \to W^+W^-,\ ZZ$$

Such processes can be described in an effective-Lagrangian approach, where higher-order corrections to the scattering amplitude are described by new parameters $\alpha_4, \alpha_5, \ldots$. `WHIZARD` defines these anomalous couplings in the model `SM_ac.mdl`.

At an e^+e^- collider, WW scattering processes occur as a subprocess of

$$e^-e^+ \to \nu_e\bar{\nu}_e q\bar{q}q\bar{q}.$$

This can be simulated in a single run, if we set up the process configuration file `whizard.prc` as follows:

```
# WHIZARD configuration file

model   SM_ac

alias q u:d
alias Q U:D
alias n n1:n2:n3
alias N N1:N2:N3

# Tag    In      Out                          Method Option
#=======================================================
# On-shell process:
ww       W+,W-   W+,W-                        omega
zz       W+,W-   Z,Z                          omega

# Full six-fermion matrix elements:
nnqqqq   e1,E1   n,N,q,Q,q,Q                  omega   c
enqqqq   e1,E1   e1,N1,q,Q,q,Q                omega   c,w:c
neqqqq   e1,E1   n1,E1,q,Q,q,Q                omega   c,w:c
eeqqqq   e1,E1   e1,E1,q,Q,q,Q                omega   c,w:c

# First neutrino generation only:
# WW and ZZ
nnuudd   e1,E1   n1,N1,u,U,d,D                omega   c
# WW only
nnucsd   e1,E1   n1,N1,u,C,s,D                omega   c
eeucsd   e1,E1   e1,E1,u,C,s,D                omega   c,w:c
# ZZ only
nnuuss   e1,E1   n1,N1,u,U,s,S                omega   c
```

```
# WZ
enudss   e1,E1   e1,N1,u,D,s,S             omega   c,w:c

# Second neutrino generation only:
# WW and ZZ
nnuudd2  e1,E1   n2,N2,u,U,d,D             omega   c
# WW only
nnucsd2  e1,E1   n2,N2,u,C,s,D             omega   c
# ZZ only
nnuuss2  e1,E1   n2,N2,u,U,s,S             omega   c
```

This is actually an abridged version of the process file `whizard.prc.ww-strong` that comes with the standard distribution.

In the chosen model, the Higgs boson is actually not absent, but its mass is set to a very large value (resp. infinity) by default. Apart from the signal process `nnqqqq` we have included some important background processes, which must be considered if the final-state electron is not observed. For these processes, gauge invariance is an issue, and we must set the `w:f` (fudged-width) or `w:c` (constant-width) option to obtain a consistent result. Note, however, that the fudged-width option does not work with the `'c'` flag for colored amplitudes, but the constant-width option works perfectly fine.

The setup below is (for historical reasons) for the original TESLA collider design. We may have polarization and have to account for ISR and beamstrahlung. The input file `whizard.in` specifies 80 % left-handed electron polarization and 40 % right-handed positron polarization:

```
&process_input
process_id = "nnqqqq"
sqrts = 800
luminosity = 0
polarized_beams = T
structured_beams = T
/

&integration_input
calls = 1 100000 15 100000 5
default_Q_cut = 0
/

&simulation_input
/

&diagnostics_input
/

&parameter_input
```

```
ms = 0
mc = 0
a4 = 0
a5 = 0
/

&beam_input
particle_name = "e-"
polarization = 0.80 0
CIRCE_on = T
CIRCE_acc = 2
ISR_on = T
ISR_alpha = 0.0072993
ISR_m_in = 0.000511
/

&beam_input
particle_name = "e+"
polarization = 0 0.40
CIRCE_on = T
CIRCE_acc = 2
ISR_on = T
ISR_alpha = 0.0072993
ISR_m_in = 0.000511
/
```

Polarization is switched on by the entry `polarized_beams` in the first block. Then, the `polarization` settings in the `beam_input` blocks are respected. Similarly, `structured_beams` switches on possible spectra and/or structure function settings. Now, the beam particles are no longer automatically defined by the process, but have to be explicitly defined for each beam.

The beamstrahlung settings (`CIRCE`) are for the accelerator type 2 (TESLA), default parameterization version and revision numbers. Concerning initial-state radiation (ISR), we should set the electromagnetic coupling constant equal to the low-energy value of 1/137 since on-shell photons are radiated. The incoming mass must be reset equal to 511 keV, since the physical electron mass `me` has been set to zero.

In the parameter section, the two anomalous couplings α_4 and α_5 are included. Here, we set them to zero which is also the default value.

The signal process considered here is quite well-behaved, so, in the integration section, we choose a number of iterations that is smaller than the default for this class of processes, to limit execution time. (The zero value set for the Q cut is unnecessary for the signal process considered here and could be left out. It is useful for integrating the background process $e^+e^- \to e^+e^-q\bar{q}q\bar{q}$. With no Q cut on the final-state electron, we get the total cross section. The parameter `default_Q_cut` has the side effect of setting the scale for the integration over the transverse momentum. If the parameter is zero, the electron mass is taken as setting the

scale, which is appropriate for the total cross section.) The actual cuts for the signal and background processes may be taken as the default ones, or we could write them explicitly in a file. The default cuts require a minimum invariant mass of 10 GeV for each quark pair.

The output shown below is for the complete process, including all possible flavor combinations in the process labeled `nnqqqq`

$$e^-e^+ \to \nu_e\bar{\nu}_e q\bar{q}q\bar{q}; \qquad \nu = \nu_e, \nu_\mu, \nu_\tau; \qquad q = u, d$$

```
 ! WHIZARD 1.97 (May 31 2011)
 ! Reading process data from file whizard.in
 ! Wrote whizard.out
 ! Reading phase space configurations from file whizard.phx
 !
 ! Process nnqqqq:
 !    e a-e -> nu_e a-nu_e   u a-u   u a-u
.....
 !    e a-e -> nu_tau a-nu_tau   d a-d   d a-d
 !  128  64 ->      1        2   4   8  16  32
 ! Process energy set to     800.00     GeV
 ! Active structure functions for beam 1:
 !   CIRCE:          e -> e                (generator)
 !   ISR:            e -> e
 ! Active structure functions for beam 2:
 !   CIRCE:          a-e -> a-e              (generator)
 !   ISR:            a-e -> a-e
 ! Warning: CIRCE: Beamstrahlung effect on polarization will be ignored.
 ! Warning: CIRCE: Beamstrahlung effect on polarization will be ignored.
 ! Warning: ISR: Effect on beam polarization will be ignored.
 ! Warning: ISR: Effect on beam polarization will be ignored.
 !   416 phase space channels found for process nnqqqq
 ! Scanning phase space channels for equivalences ...
 ! Phase space:    2160 equivalence relations found.
 ! Note: This cross section may be infinite without cuts.
.....
 ! Wrote default cut configuration file whizard.nnqqqq.cut0
 !
 ! Created grids:    416 channels, 18 dimensions with 20 bins
 ! Reading grid data from file whizard.nnqqqq.grb
 ! Reading grid data from file whizard.nnqqqq.grc
 !
 ! WHIZARD run for process nnqqqq:
 !===============================================================================
 ! It      Calls  Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
 !-------------------------------------------------------------------------------
 ! Reading cut configuration data from file whizard.cut1
 ! No cut data found for process nnqqqq
 ! Using default cuts.
 cut M of   12      within  1.00000E+01  1.00000E+99
 cut M of   20      within  1.00000E+01  1.00000E+99
 cut M of   36      within  1.00000E+01  1.00000E+99
 cut M of   24      within  1.00000E+01  1.00000E+99
```

```
cut M of   40      within  1.00000E+01  1.00000E+99
cut M of   48      within  1.00000E+01  1.00000E+99
! Using grids and results from file:
   1     100000  6.8946962E+00  1.25E+00   18.14   57.37*  0.75    0.00   1
!-------------------------------------------------------------------------------
! Using grids and results from file:
   2     100000  6.8564783E+00  1.19E+00   17.42   55.09*  0.85
   3     100000  1.2555981E+01  5.38E+00   42.87  135.55   0.45
   4     100000  6.0430895E+00  3.31E-01    5.48   17.34*  0.86
   5     100000  6.8791354E+00  5.83E-01    8.48   26.82   1.02
   6     100000  6.2190338E+00  1.86E-01    2.99    9.46*  1.16
   7     100000  6.0883991E+00  1.37E-01    2.24    7.10*  1.66
   8     100000  6.1381611E+00  1.15E-01    1.88    5.94*  1.66
   9     100000  6.1823849E+00  1.09E-01    1.77    5.59*  1.70
  10     100000  6.2165210E+00  2.23E-01    3.59   11.34   1.49
  11     100000  6.0081519E+00  8.52E-02    1.42    4.48*  2.23
  12     500000  6.0368363E+00  4.79E-02    0.79    5.62   0.73
  13     100000  5.7156812E+00  1.38E-01    2.41    7.61   3.19
  14     100000  6.0900287E+00  1.15E-01    1.88    5.95   1.75
  15     100000  6.2165952E+00  1.25E-01    2.01    6.37   1.56
!-------------------------------------------------------------------------------
! Integrating (fixed wgts.):   5 samples of     100000 calls ...
  16     500000  6.1040057E+00  5.19E-02    0.85    6.01   0.68    1.19   5
!-------------------------------------------------------------------------------
!
! Time estimate for generating 10000 unweighted events:    0h 26m 58s
!===============================================================================
! Summary (all processes):
!-------------------------------------------------------------------------------
! Process ID     Integral[fb]  Error[fb]   Err[%]        Frac[%]
!-------------------------------------------------------------------------------
  nnqqqq         6.1040057E+00  5.19E-02    0.85         100.00
!-------------------------------------------------------------------------------
  sum            6.1040057E+00  5.19E-02    0.85         100.00
!===============================================================================
! Wrote whizard.out
! Integration complete.
! No event generation requested
```

6.4 W endpoint at LHC

We want to study the Jacobian peak of the W boson at the LHC. Therefore, we introduce aliases for leptons and neutrinos as well as for light jets and use the following `whizard.prc` file:

```
# WHIZARD configuration file
# The selected model
model   SM

alias j u:U:d:D:g
```

```
alias ll e1:e2
alias neutrino n1:n2:N1:N2

# Processes
# Tag           In      Out              Method  Option
#=================================================
enj             j,j     ll,neutrino,j    omega   c
```

Note that we sum only over up and down quark (as well as the gluon) for the parton as well as for the final jet. More quarks can easily be added, but do not change the picture. We sum over the first two lepton (and neutrino) generations, but take only production of negatively charged W bosons into account. Many technical details have been discussed above and need not be repeated here.

We use the following input file:

```
&process_input
process_id = "enj"
sqrts = 14000
polarized_beams = F
structured_beams = T
/

&integration_input
calls = 3 10000 10 20000 5 100000
/

&simulation_input
n_events = 1000
/

&diagnostics_input /

&parameter_input /

&beam_input
 particle_name = "p"
 LHAPDF_on = T
 LHAPDF_file = "cteq6ll.LHpdf"
 PDF_running_scale = T
/

&beam_input
 particle_name = "p"
```

```
 LHAPDF_on = T
 LHAPDF_file = "cteq6ll.LHpdf"
 PDF_running_scale = T
/
```

So the setup is for a 14 TeV LHC (pp) collider, using `cteq6ll.LHpdf` leading order CTEQ PDFs from LHAPDF. We chose to use a running scale (i.e. $\sqrt{\hat{s}}$) rather than the W mass. Instead of choosing a specific luminosity we just generated 1,000 unweighted events. We adopt only very mild cuts, i.e. a 10 GeV p_T cut on the charged lepton and the jet in the file `whizard.cut1`:

```
process enj
cut PT of 1 within 10 99999
cut PT of 2 within 10 99999
cut PT of 4 within 10 99999
```

Starting `WHIZARD` gives the following run output:

```
! WHIZARD 1.97 (May 31 2011)
! Reading process data from file whizard.in
! Wrote whizard.out
! Reading phase space configurations from file whizard.phx
!
! Process enj:
!  a-u   d ->   e a-nu_e   g
.....
!    g   d ->  mu a-nu_mu   u
!   16   8 ->   1       2   4
! Process energy set to     14000.     GeV
************************************
*       LHAPDF Version 5.8.4       *
*   Configured for the following:  *
*             All PDFs             *
*         FULL MEMORY option       *
*    Maximum  3 concurrent set(s)  *
************************************

>>>>>> PDF description: <<<<<<
CTEQ6L1 - LO with LO alpha_s
.....
>>>>>>                    <<<<<<
Parametrization: CTEQ6

! Active structure functions for beam 2:
!   LHAPDF:         p -> a-u
!     3 phase space channels found for process enj
! Scanning phase space channels for equivalences ...
! Phase space:       3 equivalence relations found.
! Note: This cross section may be infinite without cuts.
! Wrote default cut configuration file whizard.enj.cut0
.....
```

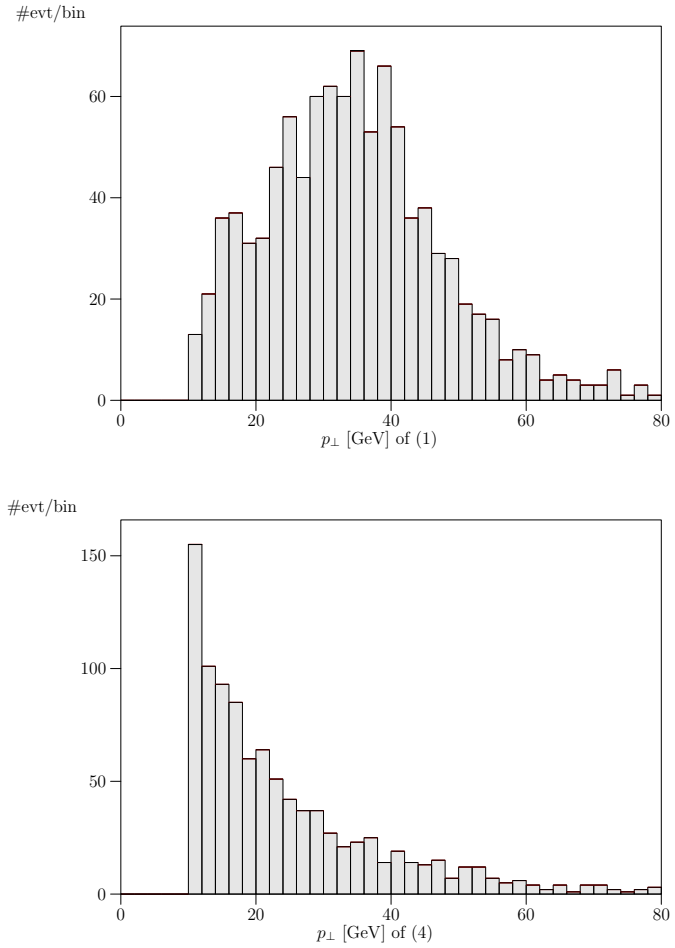


Figure 17: *Histograms for the process* $pp \to \ell^- \bar{\nu} j$ *at* $\sqrt{s} = 14$ *TeV. Top:* p_T *distribution of the lepton, Bottom:* p_T *distribution of the jet.*

```
!
! Created grids:      3 channels,  7 dimensions with 20 bins
!
! WHIZARD run for process enj:
!===============================================================================
! It      Calls  Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!-------------------------------------------------------------------------------
! Reading cut configuration data from file whizard.cut1
! Replacing default cuts by user-defined cuts.
cut pT of    1      within  1.00000E+01  9.99990E+04
cut pT of    2      within  1.00000E+01  9.99990E+04
cut pT of    4      within  1.00000E+01  9.99990E+04
! Preparing (fixed weights):   3 samples of      10000 calls ...
   1      30000  5.1065657E+06  1.32E+05    2.58    4.47*  0.21    2.66   3
!-------------------------------------------------------------------------------
! Adapting (variable wgts.):  10 samples of      20000 calls ...
   2      20000  5.2468949E+06  1.05E+05    2.01    2.84*  1.65
```

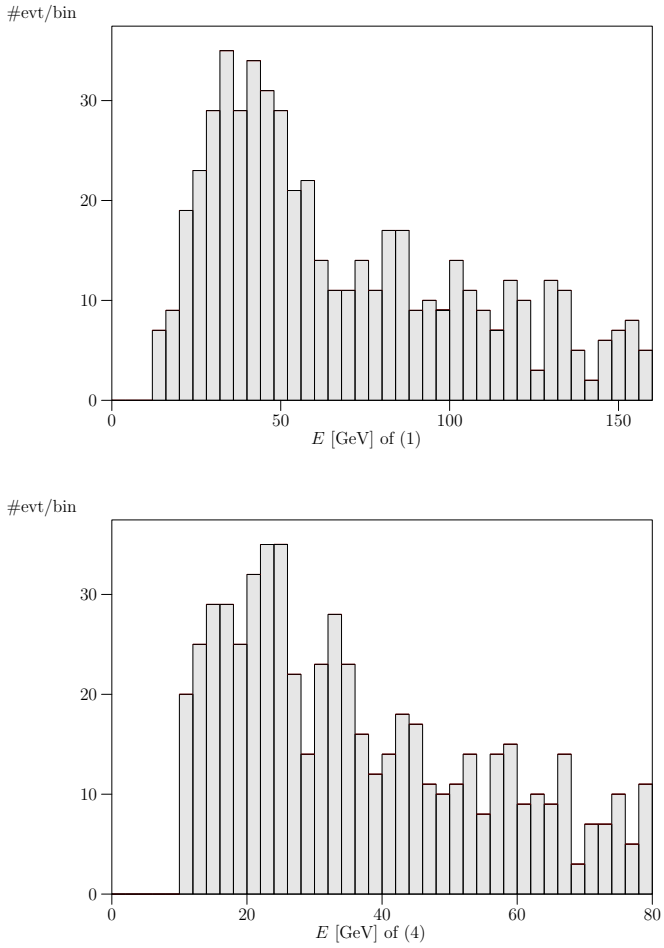


Figure 18: *Histograms for the process* $pp \to \ell^- \bar{\nu} j$ *at* $\sqrt{s} = 14$ *TeV. Top: Energy distribution of the lepton, Bottom: Energy distribution of the jet.*

```
   3      20000  5.1653531E+06  5.66E+04    1.10    1.55*  3.25
   4      20000  5.2198528E+06  4.94E+04    0.95    1.34*  3.14
   5      20000  5.1720114E+06  5.11E+04    0.99    1.40   2.18
   6      20000  5.1487802E+06  4.58E+04    0.89    1.26*  7.61
   7      20000  5.1670713E+06  4.32E+04    0.84    1.18*  3.95
   8      20000  5.2170596E+06  4.51E+04    0.86    1.22   3.01
   9      20000  5.2760343E+06  4.92E+04    0.93    1.32   2.50
  10      20000  5.2178545E+06  4.38E+04    0.84    1.19   6.15
  11      20000  5.2588972E+06  4.46E+04    0.85    1.20   4.02
!-------------------------------------------------------------------------------
! Integrating (fixed wgts.):   5 samples of     100000 calls ...
  12     500000  5.2309417E+06  9.60E+03    0.18    1.30   0.66    0.34   5
!-------------------------------------------------------------------------------
!
! Time estimate for generating 10000 unweighted events:    0h 00m 44s
!===============================================================================
! Summary (all processes):
```

```
!------------------------------------------------------------------------------
! Process ID     Integral[fb]  Error[fb]   Err[%]        Frac[%]
!------------------------------------------------------------------------------
  enj            5.2309417E+06  9.60E+03    0.18         100.00
!------------------------------------------------------------------------------
  sum            5.2309417E+06  9.60E+03    0.18         100.00
!==============================================================================
! Wrote whizard.out
! Integration complete.
!
! Reading analysis configuration data from file whizard.cut5
! Found   1 analysis configuration dataset.
! Event sample corresponds to luminosity [fb-1] =  0.1912E-03
! Event sample corresponds to      150712  weighted events
! Generating       1000 unweighted events ...
! Event generation finished.
!==============================================================================
! Analysis results for process enj:
! It      Events Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!------------------------------------------------------------------------------
  13       1000  5.2309417E+06  1.65E+05    3.16    1.00 100.00
!------------------------------------------------------------------------------
! Warning: Excess events:    5.0       (    0.50% )  | Maximal weight:  3.41
```

For the analysis, we want to histogram the p_T distributions for the charged lepton and the jet as well as their corresponding energies. This is achieved with the analysis file `whizard.cut5`:

```
process enj
histogram PT of 1 within 0 80 nbin 40
histogram PT of 4 within 0 80 nbin 40
histogram E of 1 within 0 160 nbin 40
histogram E of 4 within 0 80 nbin 40
```

In the following figures we show the p_T distribution of the lepton which shows the Jacobian peak from the W decay, while the corresponding histogram of the jet does not show this feature 17. Fig. 18 shows the energy of the lepton and the jet, respectively. These are more less resembling the behavior of the p_T distribution.

6.5 Z' in Drell-Yan Production at the LHC

As a second example for LHC physics, we consider the simplest beyond the Standard Model (BSM) search, namely that for Z' bosons in Drell-Yan production with subsequent decay into electrons. After having defined the process in `whizard.prc`:

```
# WHIZARD configuration file
# The selected model
model   Zprime
alias p u:d:s:c:U:D:S:C
# Processes
# Tag           In      Out             Method  Option
#=================================================
drellyan        p,p     e1,E1           omega   c
```

Note that we have to set the model to `Zprime`, otherwise for the SM `WHIZARD` would generate ordinary Drell-Yan production and our search for a high-energetic dilepton resonance would be in vain.

We impose the following cuts on the process: basically experimental trigger cuts, demanding the final state leptons to have a transverse momentum $p_T > 50\,\mathrm{GeV}$ and to stay centrally in the detector, i.e. $-2.5 < \eta < +2.5$. Furthermore, we concentrate on the discovery region for a heavy vector resonance in Drell-Yan production, meaning that we look only for dilepton invariant masses larger than 800 GeV (below we will assume that the Z' boson has a mass of 1.5 TeV). The cuts are set by the following statements in the file `whizard.cut1`:

```
process drellyan
cut PT of 1 within 50 99999
cut PT of 2 within 50 99999
cut Eta of 1 within -2.5 2.5
cut Eta of 2 within -2.5 2.5
cut M of 3 within 800 99999
```

The input file is only a slight modification of the one `whizard.in.Zprime` that can be found in the `conf` subdirectory:

```
&process_input
process_id = "drellyan"
sqrts = 14000
luminosity = 0
polarized_beams = F
structured_beams = T
/

&integration_input
calls = 1 10000 10 20000 5 100000
/

&simulation_input
```

```
 n_events = 100000
 /

 &diagnostics_input /

 &parameter_input
 Me = 0
 Ms = 0
 Mc = 0
 MH = 115
 wH = 0.3228E-02
 MZH = 1500
 wZH = 50
!!! Zprime couplings as multiples of Z couplings
 v_lep = 1.0
 v_neu = 1.0
 v_up  = 1.0
 v_dwn = 1.0
 a_lep = 1.0
 a_neu = 1.0
 a_up  = 1.0
 a_dwn = 1.0
!!! Explicit Zprime couplings
 glepv = 0.0
 gneuv = 0.0
 gupv  = 0.0
 gdwnv = 0.0
 glepa = 0.0
 gneua = 0.0
 gupa  = 0.0
 gdwna = 0.0
 /

 &beam_input
 particle_name = "p"
 LHAPDF_on = T
 LHAPDF_file = "cteq6ll.LHpdf"
 LHAPDF_set = 0
 !!! Should be chosen equal to MZH
 PDF_scale = 1500
 /

 &beam_input
 particle_name = "p"
 LHAPDF_on = T
 LHAPDF_file = "cteq6ll.LHpdf"
 LHAPDF_set = 0
 !!! Should be chosen equal to MZH
 PDF_scale = 1500
 /
```

Again, `cteq6ll.LHpdf` sets from `LHAPDF` are used, this time with a scale equal to the mass of the Z' boson, namely 1.5 TeV. The width is not calculated, but set explicitly to 50 GeV. For the couplings, there are several parameters which could be set in that particular BSM model. The couplings `glepv` etc. allow for explicit values for the Z' couplings to the SM fermions, while `v_lep`, `a_lep` etc. are just rescalings of the SM Z boson couplings to the fermions. As these values are set equal to one everywhere, this Z' is the infamous strawman model of a sequential Z' which however from a theoretical point of view is useless, because it is a non-renormalizable model.

Starting `WHIZARD` yields the following output:

```
 ! WHIZARD 1.97 (May 31 2011)
 ! Reading process data from file whizard.in
 ! Wrote whizard.out
 !
 ! Process drellyan:
 !    u a-u ->   e a-e
.....
 !    8   4 ->   1   2
 ! Process energy set to     14000.     GeV
 ************************************
 *       LHAPDF Version 5.8.3       *
 *   Configured for the following:  *
 *             All PDFs             *
 *         FULL MEMORY option       *
 *    Maximum  3 concurrent set(s)  *
 ************************************

 >>>>>> PDF description: <<<<<<
.....
 >>>>>>                   <<<<<<

 Parametrization: CTEQ6

 ! Active structure functions for beam 2:
 !   LHAPDF:         p -> a-u
 ! Reading vertices from file whizard.mdl ...
 ! Model file:       62 trilinear vertices found.
 ! Model file:       62 vertices usable for phase space setup.
 ! Generating phase space channels for process drellyan...
 ! Phase space:       2 phase space channels generated.
 ! Scanning phase space channels for equivalences ...
 ! Phase space:       2 equivalence relations found.
 ! Note: This cross section may be infinite without cuts.
 ! Wrote default cut configuration file whizard.drellyan.cut0
.....
 ! Wrote phase space configurations to file whizard.phx
 !
 ! Created grids:      2 channels,  4 dimensions with 20 bins
 !
 ! WHIZARD run for process drellyan:
 !===============================================================================
```

```
! It      Calls  Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!------------------------------------------------------------------------------
! Reading cut configuration data from file whizard.cut1
! Replacing default cuts by user-defined cuts.
cut pT of    1      within  5.00000E+01  9.99990E+04
cut pT of    2      within  5.00000E+01  9.99990E+04
cut eta of    1      within -2.50000E+00  2.50000E+00
cut eta of    2      within -2.50000E+00  2.50000E+00
cut M of    3      within  8.00000E+02  9.99990E+04
! Preparing (fixed weights):   1 sample of      10000 calls ...
   1      10000  6.3659147E+01  7.91E+00   12.42   12.42*  0.22    0.00   1
!------------------------------------------------------------------------------
! Adapting (variable wgts.):  10 samples of      20000 calls ...
   2      20000  8.2530400E+01  6.53E+00    7.91   11.19*  0.26
   3      20000  6.6421810E+01  1.83E+00    2.75    3.89*  1.88
   4      20000  6.9202182E+01  8.07E-01    1.17    1.65*  7.94
   5      20000  6.9246385E+01  4.22E-01    0.61    0.86*  4.39
   6      20000  6.9431259E+01  3.46E-01    0.50    0.71* 11.81
   7      20000  6.9932020E+01  3.67E-01    0.52    0.74   9.65
   8      20000  6.9123994E+01  3.80E-01    0.55    0.78   7.37
   9      20000  6.9979530E+01  3.95E-01    0.56    0.80   8.59
  10      20000  7.0999763E+01  3.80E-01    0.54    0.76   8.86
  11      20000  7.0029224E+01  4.01E-01    0.57    0.81   9.40
!------------------------------------------------------------------------------
! Integrating (fixed wgts.):   5 samples of     100000 calls ...
  12     500000  6.9702626E+01  7.24E-02    0.10    0.73   3.28    0.51   5
!------------------------------------------------------------------------------
!
! Time estimate for generating 10000 unweighted events:    0h 00m 05s
!==============================================================================
! Summary (all processes):
!------------------------------------------------------------------------------
! Process ID     Integral[fb]  Error[fb]   Err[%]        Frac[%]
!------------------------------------------------------------------------------
  drellyan       6.9702626E+01  7.24E-02    0.10         100.00
!------------------------------------------------------------------------------
  sum            6.9702626E+01  7.24E-02    0.10         100.00
!==============================================================================
! Wrote whizard.out
! Integration complete.
!
! Reading analysis configuration data from file whizard.cut5
! Found   1 analysis configuration dataset.
! Event sample corresponds to luminosity [fb-1] =   1435.
! Event sample corresponds to     3050955  weighted events
! Generating     100000 unweighted events ...
! Event generation finished.
!==============================================================================
! Analysis results for process drellyan:
! It      Events Integral[fb]  Error[fb]   Err[%]    Acc  Eff[%]   Chi2 N[It]
!------------------------------------------------------------------------------
  13     100000  6.9702626E+01  2.20E-01    0.32    1.00 100.00
```

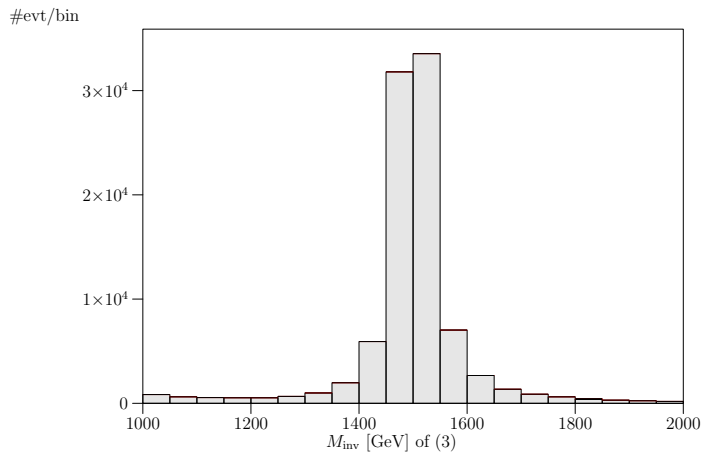


Figure 19: *Drell-Yan production of a 1.5 TeV Z' boson. Dilepton invariant mass.*

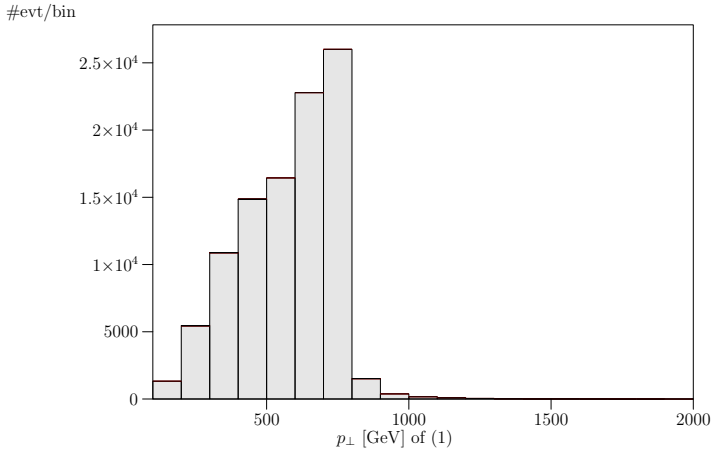


Figure 20: *Drell-Yan production of a 1.5 TeV Z' boson. p_T distribution of the negatively charged lepton.*

```
!-------------------------------------------------------------------------------
! Warning: Excess events:   42.0       (    0.04% )  | Maximal weight:  2.44
```

In the analysis, we want to investigate the dilepton invariant mass spectrum in the high-energy region, the p_T distribution of the lepton, as well as the lepton's angular correlation. Hence, we use the following analysis file `whizard.cut5`:

```
process drellyan
histogram M of 3 within 1000 2000 step 50
histogram PT of 1 within 100 2000 step 100
histogram AAD of 1 within 0 180 step 10
```

The three distributions are shown in the three Figs. 19, 20, and 21.

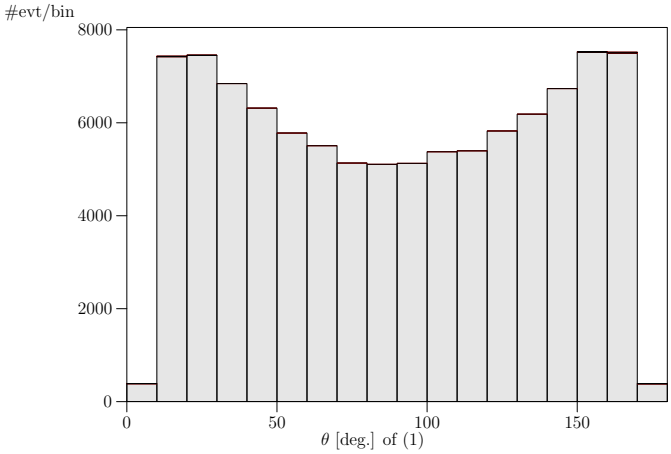


Figure 21: *Drell-Yan production of a 1.5 TeV Z' boson. Angle of the negatively charged lepton with respect to the beam.*

6.6 Energy and more general parameter scans

If one wants to scan over a parameter range (either the center-of-mass energy for which there is also the option to use a uniform beam spectrum or another parameter of the model), there is the shell script `scan_range.pl` which can be found in the `conf` subdirectory:

```
 #!/usr/bin/perl
# Loop over energy and extract results

$prog = "./whizard";
$file = "whizard.out";

$Emin = 500;
$Emax = 5000;
$Estep = 50;

@processes = (
              "eemm"
              );

foreach $proc (@processes) {
    $proc{$proc} = 1;
    eval "\$tab_".$proc." .= 'Cross section for process $proc\n'";
    eval "\$tab_".$proc." .= '# sqrts[GeV]  sigma[fb]         error[fb]\n'";
}

for ($E=$Emin; $E <= $Emax; $E=$E+$Estep) {
    system ("$prog -P'Mzh=$E'");
#    system ("$prog -p'sqrts=$E'");
    open (RESULT, "$file") or die;
    while (<RESULT>) {
        ($proc, $cs, $err) = split;
        if ($proc{$proc}) {
            eval "\$tab_".$proc." .= '    $E      $cs      $err\n'";
        }
    }
    close (RESULT);
}

open (OUT, ">scan_range.dat");
foreach $proc (@processes) {
    eval "print OUT \$tab_".$proc;
    print OUT "\n\n";
}
close (OUT);
```

This `PERL` script takes a list of processes (here only `eemm` intended to be $e^-e^+ \to \mu^-\mu^+$) to be attached to the `PERL` array `@processes`. Then, there is one input parameter `$E` that is varied between the two values `$Emin` and `$Emax` with steps `$Estep`. The following line of code `system ("$prog -P'Mzh=$E'")` calls the program (i.e. `WHIZARD`) with the command line option setting the parameter (here `Mzh`, i.e. the mass of the heavy Z' boson and varies it. The

center-of-mass energy must then be set explicitly in the `whizard.in` file. (If the parameter to be varied is $\sqrt{s}$ itself, the corresponding entry in the input file could be empty). The `PERL` script produces an output file `scan_rannge.dat` which contains histogrammed data for the corresponding variable. Note that `WHIZARD` 2 has an intrinsic option for parameter scans, and is able to directly produce plain histogram files. For more information confer the `WHIZARD` 2 manual.

Acknowledgements

We would like to thank R. Chierici, K. Desch, N. Meyer and S. Rosati, who used preliminary or early versions of the program for real-life applications and thus helped a lot in debugging and improving code. For a complete list of people who helped us with suggestions, advice and debugging etc. confer the `WHIZARD` 2 manual's acknowledgements. There is also the most recent list of funding agencies which supported this project.

References

[1] T. Sjöstrand, Comput. Phys. Commun. **82** (1994) 74.

[2] A. Pukhov, *et al.*, Preprint INP MSU 98-41/542, `hep-ph/9908288`.

[3] T. Stelzer and W.F. Long, Comput. Phys. Commun. **81** (1994) 357.

[4] T. Ohl, *Proceedings of the Seventh International Workshop on Advanced Computing and Analysis Technics in Physics Research*, ACAT 2000, Fermilab, October 2000, IKDA-2000-30, `hep-ph/0011243`; M. Moretti, Th. Ohl, and J. Reuter, LC-TOOL-2001-040

[5] T. Ohl, Comput. Phys. Commun. **120** (1999) 13.

[6] T. Ohl, Comput. Phys. Commun. **101** (1997) 269.

[7] M. Skrzypek and S. Jadach, Z. Phys. **C49** (1991) 577.

[8] A. Djouadi, J. Kalinowski, M. Spira, Comput. Phys. Commun. **108** (1998) 56-74.

[9] E. Boos *et al.*, in: Proc. Les Houches 2001, `hep-ph/0109068`

[10] P. Z. Skands *et al.*, JHEP **0407**, 036 (2004) [arXiv:hep-ph/0311123].

[11] B. C. Allanach *et al.*, Comput. Phys. Commun. **180**, 8 (2009) [arXiv:0801.0045 [hep-ph]].

[12] K. Hagiwara *et al.*, Phys. Rev. D **73**, 055005 (2006) [arXiv:hep-ph/0512260].

[13] A. Alboteanu, W. Kilian and J. Reuter, JHEP **0811**, 010 (2008) [arXiv:0806.4145 [hep-ph]].

[14] B. C. Allanach *et al.*, in *Proc. of the APS/DPF/DPB Summer Study on the Future of Particle Physics (Snowmass 2001)* ed. N. Graf, Eur. Phys. J. C **25** (2002) 113 [eConf **C010630** (2001) P125] [arXiv:hep-ph/0202233].

[15] J. A. Aguilar-Saavedra *et al.*, Eur. Phys. J. C **46**, 43 (2006) [arXiv:hep-ph/0511344].

[16] J. Reuter and F. Braam, AIP Conf. Proc. **1200**, 470 (2010) [arXiv:0909.3059 [hep-ph]].

[17] N. D. Christensen *et al.*, Eur. Phys. J. C **71**, 1541 (2011) [arXiv:0906.2474 [hep-ph]].

[18] J. Alwall *et al.*, Comput. Phys. Commun. **176**, 300 (2007) [arXiv:hep-ph/0609017].

[19] T. Binoth, N. Greiner, A. Guffanti, J. Reuter, J. P. Guillet and T. Reiter, Phys. Lett. B **685**, 293 (2010) [arXiv:0910.4379 [hep-ph]].

[20] N. Greiner, A. Guffanti, T. Reiter and J. Reuter, arXiv:1105.3624 [hep-ph].