

MC-TESTER v. 1.23: a universal tool for comparisons of Monte Carlo predictions for particle decays in high energy physics [†]

N. Davidson^{a,d}, P. Golonka^b, T. Przedziński^c, Z. Wąs^{d,e}

^a *University of Melbourne, Department of Physics
Australia.*

^b *CERN, IT/CO-BE, CH-1211 Geneva 23, Switzerland.*

^c *The Faculty of Physics, Astronomy and Applied Computer Science,
Jagellonian University, Reymonta 4, 30-059 Cracow, Poland.*

^d *Institute of Nuclear Physics, Radzikowskiego 152, 31-342 Cracow, Poland.*

^e *CERN PH-TH, CH-1211 Geneva 23, Switzerland.*

Abstract

Theoretical predictions in high energy physics are routinely provided in the form of Monte Carlo generators. Comparisons of predictions from different programs and/or different initialization set-ups are often necessary. MC-TESTER can be used for such tests of decays of intermediate states (particles or resonances) in a semi-automated way.

Since 2002 new functionalities were introduced into the package. In particular, it now works with the HepMC event record, the standard for C++ programs. The complete set-up for benchmarking the interfaces, such as interface between τ -lepton production and decay, including QED bremsstrahlung effects is shown. The example is chosen to illustrate the new options introduced into the program. From the technical perspective, our paper documents software updates and supplements previous documentation.

As in the past, our test consists of two steps. Distinct Monte Carlo programs are run separately; events with decays of a chosen particle are searched, and information is stored by MC-TESTER. Then, at the analysis step, information from a pair of runs may be compared and represented in the form of tables and plots.

Updates introduced in the program up to version 1.24.4 are also documented. In particular, new configuration scripts or script to combine results from multitude of runs into single information file to be used in analysis step are explained.

To be published in Computer Physics Communications,

CERN-LCGAPP-2008-02
IFJPAN-IV-2008-9

[†] This work is partially supported by EU Marie Curie Research Training Network grant under the contract No. MRTN-CT-2006-0355505 and by Polish Government grant N202 06434 (2008-2010).

Contents

1	Introduction	4
2	Installation and generation step (update for ref. [1])	5
2.1	Examples of C++ generation step	6
2.1.1	Tau decays from PYTHIA 8.1	6
2.1.2	B decays from EvtGenLHC	7
3	Analysis (update for ref. [1])	8
3.1	Running the analysis step from an external directory	8
3.2	New options in graphical representation of histograms	8
4	Package organization (update for ref. [1])	11
4.1	Directory tree	11
4.2	Libraries	12
4.3	Format and syntax of the SETUP.C file	12
4.4	How to make MC-TESTER run with other generators	12
4.4.1	The case of C++	12
4.5	How to make MC-TESTER run with on-flight modified event records	14
5	The use of lists in MC-TESTER	14
6	Example of advanced MC-TESTER use: benchmarks for spin correlations in heavy object decays.	17
6.1	Default UserTreeAnalysis	17
7	Outlook	18
A	Appendix: MC-TESTER setup and input parameters (update for ref. [1])	21
A.1	Definition of parameters in the SETUP.C file	21
A.1.1	Setup::UserTreeAnalysis	21
A.1.2	Setup::mass_power	22
A.1.3	Setup::mass_scale_on	22
A.1.4	Setup::use_log_y	22
A.1.5	Setup::rebin_factor	23
A.2	C++ configuration of MC-TESTER	23
B	Appendix: updates from version 1.23 up to version 1.24.4	23
B.1	Changes introduced in version 1.24.2	23
B.2	LCG configuration scripts; available from version 1.24.2	24
B.3	Merging MC-TESTER output files; available from version 1.24.3	25
B.4	Updates introduced in version 1.24.4	26

PROGRAM UPDATE SUMMARY

Title of the program: MC-TESTER, version 1.23 and version 1.24.4

Tested on various platforms and operating systems: Linux SLC 4.6 and SLC 5, Fedora 8, Ubuntu 8.2 etc.

Programming languages used: C++, FORTRAN77

Tested and compiled with: gcc 3.4.6, 4.2.4 and 4.3.2 with g77/gfortran

Size of the package:

23.4 MB directory including example programs (6.5 MB compressed distribution archive), without ROOT libraries.

Additional disk space required:

Depends on the analyzed particle: 14.4 MB in the case of τ lepton decays (30 decay channels, 594 histograms, 73-pages booklet).

Keywords:

particle physics, decay simulation, Monte Carlo methods, invariant mass distributions, programs comparison

Nature of the physical problem:

The decays of individual particles are well defined modules of a typical Monte Carlo program chain in high energy physics. A fast, semi-automatic way of comparing results from different programs is often desirable for the development of new programs, in order to check correctness of the installations or for discussion of uncertainties.

Method of solution:

A typical HEP Monte Carlo program stores the generated events in event records such as HepMC, HEPEVT or PYJETS. MC-TESTER scans, event by event, the contents of the record and searches for the decays of the particle under study. The list of the found decay modes is successively incremented and histograms of all invariant masses which can be calculated from the momenta of the particle decay products are defined and filled. The outputs from the two runs of distinct programs can be later compared. A booklet of comparisons is created: for every decay channel, all histograms present in the two outputs are plotted and parameter quantifying shape difference is calculated. Its maximum over every decay channel is printed in the summary table.

Restrictions on the complexity of the problem: Only first 200 decay channels that were found will initialize histograms and if the multiplicity of decay products in a given channel was larger than 7, histograms will not be created for that channel as well.

Typical running time:

Varies substantially with the analyzed decay particle, but generally speed estimation of the old version remain valid. On a PC/Linux with 2.0 GHz processors MC-TESTER increases the run time of the τ -lepton Monte Carlo program TAUOLA by 4.0 seconds for every 100 000 analyzed events (generation itself takes 26 seconds). The analysis step takes 13 seconds; L^AT_EX processing takes additionally 10 seconds. Generation step runs may be executed simultaneously on multi-processor machines.

New features: HepMC interface, use of lists in definition of histograms and decay channels, filters for decay products or secondary decays to be omitted, bug fixing, extended flexibility in representation of program output, installation configuration scripts, merging multiple output

files from separate generations.

Accessibility:

web page: <http://mc-tester.web.cern.ch/MC-TESTER/>

e-mails: Piotr.Golonka@CERN.CH,
Zbigniew.Was@CERN.CH,
tomasz.przedzinski@uj.edu.pl,
Nadia.Davidson@CERN.CH.

Reference to the program previous version:

P. Golonka, T. Pierzchała, Z. Wąs, Comput. Phys. Commun., **157**(2004) 1

1 Introduction

In the phenomenology of high-energy physics, it is important to establish uncertainties for theoretical predictions which are used in the interpretation of the experimental data is of high importance. Theoretical predictions need to be presented in the form of Monte Carlo event generators; all detector effects can therefore be easily combined with the theoretical ones, using event rejection or reweighting methods. Whenever possible, theoretical predictions are separated into individual building blocks, which are later combined into complicated Monte Carlo generator systems for the complete predictions.

A good example of such a building block is generator `TAUOLA` [2–4] for the simulation of τ lepton decay. In practical applications such a program needs to be combined with other generators for the τ lepton production, `TAUOLA universal interface` [5,6] can be then used. Additional complications arise due to other effects such as final state bremsstrahlung, `PHOTOS Monte Carlo` [7–10] can be then used.

For the purpose of benchmarking our projects, we had to design and maintain tests. Some of those tests gradually evolved into the new version of `MC-TESTER` [1] presented here. The principle of these tests is rather simple. After generation of each event by a given Monte Carlo system, the content is searched for the decay of the particle to be studied. Once found, the appropriate data is collected and stored in the form of automatically created histograms and tables.

Originally the purpose of the paper was to document `MC-TESTER` version 1.23. Recent improvements, for versions up to 1.24.4, are described now as well (see Appendix B.4). For the properties of the program existing in even older versions, we address the reader to ref. [1]. We will assume that the reader is familiar with that paper, otherwise technical aspects of the program explained here, may be difficult to follow.

For the convenience of readers interested in technical aspects of the update we keep orders of the first chapters as in `MC-TESTER`'s first documentation [1]:

- Section 2 explains updates introduced to the first (generation) step of the program. Comments on the installation procedure are also given here. In Section 3 modifications introduced in the analysis, the second step of `MC-TESTER` operation, are explained.
- Section 4 is devoted to the description of the package update. In particular, directory organization and technical information on its use; further details and explanation of input parameters may be found in the appendix A.1. Sections 4.4.1, 4.5 and Appendix A.2 are devoted to the extension of `MC-TESTER` to `HepMC` and `C++` applications.
- Section 5 is devoted to the use of lists in the algorithms responsible for defining histograms and decay channels.
- New options and examples of how to obtain refined numerical results with `MC-TESTER` are explained in Section 6.
- Section 7 closes the documentation with a discussion of the package limitations and possible future extensions.

- Changes in configuration scripts and script to merge several MC-TESTER output files introduced respectively for version 1.24.2 and 1.24.4 are documented in Appendix B.

2 Installation and generation step (update for ref. [1])

MC-TESTER is distributed in a form of an archive containing source files. Currently only the Linux and Mac OS¹ operating systems are supported: other systems may be supported in the future if sufficient interest is found. We have checked MC-TESTER on various platforms such as Scientific Linux SLC 4.6 or Ubuntu 7.10, 8.04 .

In order to run MC-TESTER the following software is needed:

- gcc² compiler suite with g++ and g77/gfortran installed.
- ROOT package properly installed and set up (please refer to [11] or ROOT_INSTALL file in doc/ subdirectory for details),
- L^AT_EX package,
- Version 1.23 requires that environmental variables ROOTSYS and LD_LIBRARY_PATH, (for some applications also HEPMCLOCATION) are properly set. It is optional to set those variables manually for version 1.24.4. The new installation procedure is explained in Appendix B.

One compiles MC-TESTER libraries using the make command to be executed in its main directory³. If completed successfully, the user is instructed on how to proceed with the example tests⁴. Examples for MC-TESTER use, based on the τ decay generators TAUOLA and PYTHIA are distributed together with the package; they reside in the examples-F77/ subdirectory.

MC-TESTER distribution is a complete, ready-to-use testing environment, with subdirectories dedicated to generation and analysis steps (see Section 4.1 for details), and run-time parameters controlled by simple configuration files (SETUP.C - see Section 4.3 and the Appendix A). The SETUP.C file needs to be put in the directory from which the generation program is being executed (usually it is the same directory in which the binary executable file exist). Examples of SETUP.C files are already present in example generation directories: they are used to set some parameters and also to note the name and details of the generator being run.

The output data file is usually put in the directory in which the generation program was executed. The name of the file and the path may however be changed using SETUP.C.

The issue of using MC-TESTER with “any” Monte Carlo generators is addressed in Section 4.4. We want to stress, that it is relatively easy to use MC-TESTER with a Monte Carlo event

¹For this case LCG configuration scripts explained in Appendix B have to be used.

²MC-TESTER has been tested in particular with gcc 3.2, 4.03, 4.1.2 and 4.3.2

³The MC-TESTER version 1.23 is set to be compiled using g77 compiler. For gfortran compiler, logical link make.inc has to be pointed (in directory platform) to make-gfortran41.inc. The 1.24.4 version does not require manual compiler setup.

⁴Also, how to prepare additional libraries to be loaded into user programs.

generator: it is sufficient to link the MC-TESTER libraries, the ROOT libraries and to insert three subroutine calls into the user's code: for MC-TESTER initialization, finalization and analysis.

For the users interested in trying only the analysis part of MC-TESTER (Section 3), and to avoid a lengthy generation phase, ready-to-use data files are provided in the directory `examples-F77/pre-generated/`. There, the MC-TESTER's `mc-tester.root` files (produced by long runs with TAUOLA and PYTHIA), are stored. To copy the files to the directories of the analysis step, the command `make move` can be used. In principle these files can be used as a reference for benchmark purposes too.

2.1 Examples of C++ generation step

Demonstrations of MC-TESTER's usage with C++ generation programs can be found in the subdirectory `example-C++/`. This includes an example of τ decay analysis for PYTHIA 8.1 [12] (using C++ and the HepMC [13] standard) and an example of B meson decay for EvtGenLHC [14] (using C++ and the HEPEVT standard). The examples are chosen, to demonstrate the program use for packages of widespread popularity.

2.1.1 Tau decays from PYTHIA 8.1

An example for the PYTHIA 8.1 event generator is given in directory `examples-C++/pythia/`. 10,000 $e^+e^- \rightarrow Z^0 \rightarrow \tau^+\tau^-$ events are generated and the decay of taus are analyzed by MC-TESTER. The output, `mc-tester.root`, contains results of the processed events and includes a number of histograms which can be compared to similar output from other Monte-Carlo generators. Configuration of the tool is done via the `examples-C++/pythia/SETUP.C` file.

To run the example, the packages PYTHIA 8.1 and HepMC version 2 have to be installed⁵. PYTHIA 8.1 has to be compiled with HepMC and the PYTHIA library `libhepmcinterface` must exist. To run the example with MC-TESTER version 1.23:

- Set environment variable `HEPMCLOCATION` to the base of HepMC's `include/` and `lib/` directories
- Set `PYTHIA_INSTALL_LOCATION` to the base of PYTHIA 8's `include/` and `lib/` directories
- The `PYTHIA8DATA` should point to directory containing PYTHIA xml documents. Generally these can be found in `$(PYTHIA_INSTALL_LOCATION)/xml/doc`.
- Compile the interface library by executing `make libHepMCEvent` in the base directory of MC-TESTER.
- Compile the example by executing `make` in `examples-C++/pythia` subdirectory.

In case of MC-TESTER version 1.24.4:

⁵This example has been tested with PYTHIA version 8.100 and HepMC versions 2.01.08 - 2.05.00. We assume that the reader is familiar with these packages and their documentation.

- Provide the location of HepMC during configuration step
- Compile the MC-TESTER libraries.
- Configure example in examples-C++/pythia; provide path to PYTHIA 8.1.
- Compile with make command.

For details of HepMC and PYTHIA 8.1 see [12, 13]. For details regarding the configuration procedures see Appendix B.

In order to run the example enter examples-C++/pythia directory and execute:

```
./pythiatest.exe
make move1 (or make move2)
```

The final step moves the output file, mc-tester.root, to the directory /analyze/prod1 (/analyze/prod2) ready for the analysis step. A second output should be produced using (preferably) different Monte Carlo generator (for example from PYTHIA 6.4 generation) and moved to /analyze/prod2.

2.1.2 B decays from EvtGenLHC

An example of the analysis of 10,000 B^+ decays can be found in the sub-directory examples-C++/evtgenlhc/. This example requires MC-TESTER to be linked with the libraries of EvtGenLHC, PYTHIA, PHOTOS, CLHEP and StdHep⁶. The path to each must be set with the following environmental variables:

- EVTGEN_INSTALL_LOCATION
- PYTHIA6_INSTALL_LOCATION
- PHOTOS_INSTALL_LOCATION
- CLHEP_INSTALL_LOCATION
- CERNLIBS_INSTALL_LOCATION

The EvtGenLHC example should be run in the examples-C++/evtgenlhc/ directory with the following commands:

```
make
./evtgen_test.exe
```

⁶This example has been tested with EvtGenLHC version 5.15, PYTHIA version 6.227.2, PHOTOS version 215.5, CLHEP version 1.9.3.1 and CERNLIBS 2006 on afs at CERN. See web pages of LHC Computing Grid Project Generator Services Subproject <http://lcgapp.cern.ch/project/simu/generator/>, LCG Savannah <https://savannah.cern.ch/projects/clhep/> and web page for Scientific Linux Installation at main CERN cluster <http://plus.web.cern.ch/plus/SLC4.html>

make move1 (or make move2)

The B^+ meson decays need to be generated with the help of another Monte Carlo generator to be compared to the results of EvtGenLHC. For that purpose our example `examples-C++/pythia/pythia_test_B.cc` can be used. The resulting booklet is particularly large, confirming technical robustness of MC-TESTER.

3 Analysis (update for ref. [1])

Data files `mc-tester.root`, referred to in the previous section are used to produce a booklet - a final results of the MC-TESTER executions. For this purpose the directory `analyze/` is prepared. No significant changes in the analysis step of MC-TESTER were introduced since its first public release. The original documentation, given in [1] is to a large degree up to date. In the following subsections we present minor changes which were nonetheless introduced.

3.1 Running the analysis step from an external directory

In some cases, for example when our program is installed centrally by an administrator, users may not have write permission for the `analyze/` directory of MC-TESTER. Therefore the analysis step will need to be executed from a directory outside MC-TESTER. The bash script `analyze/compare.sh` demonstrates how this can be done. It should be copied to the working directory, and then edited. The following need to be set:

- `FILE1`: The name of the output file from the first generator (eg. `mc-tester.root`). The path should be given relative to the current working directory.
- `FILE2`: The name of the output file from the second generator.
- `MCTESTER_DIR`: The path to MC-TESTER

MC-TESTER can be configured by a `SETUP.C` file in the working directory (See Section 4.3). If this file is not present, the default settings from `analyze/SETUP.C` are used.

Execution of `compare.sh` will produce an analysis booklet in pdf format called `tester.pdf`. Other files and directories produced during the analysis step, for example ROOT histograms, can also be found in the user's current working directory.

3.2 New options in graphical representation of histograms

Graphical representation of the plots, as present already in the original version of MC-TESTER is adequate for tests, if agreement between the two compared Monte Carlo samples confirms. Generally it is however not the case. The appropriate graphical representation of tests can be then helpful to understand the origin or nature of differences. The present version of MC-TESTER introduces a few additional options that make histograms more readable in specific cases.

The logarithmic scale option (see: A.1.4) can be activated at the analysis step of MC-TESTER. A nice illustration of this functionality is the validation plot for PHOTOS in the case of QED radiation from $W^+ \rightarrow \mu^+ \nu_\mu$ final state, see fig. 1a. The compared distributions⁷ are visible only

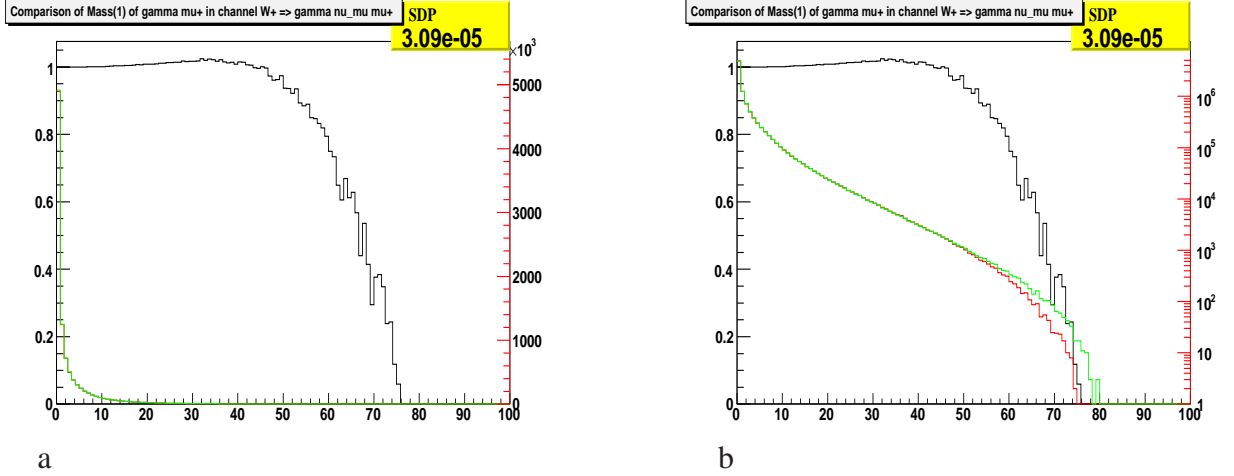


Figure 1: An example of a plot used for testing the full matrix element kernel of the $W^+ \rightarrow \mu^+ \nu_\mu \gamma$ channel. The invariant mass distribution of the $\mu^+ \gamma$ pair is peaked on the left side, making the distributions unreadable. In particular, the difference between the red and green lines can not be seen at all. Only the ratio of the compared distribution is of some use. A theoretically unprepared reader may get the impression that differences are large. If logarithmic scale is used, see fig. b, invariant mass distributions are visible over the whole spectrum and the difference is clearly localized in a region containing less than 10^{-3} of the whole sample. Calculation of SDP is not affected by the rescaling of the histograms.

in the first few bins of the histogram. If a logarithmic scale is used (see fig. 1b), one can see that the distribution extends over all kinematically allowed spectrum. Note that the ratio (thick black line) is still in linear scale marked on the left side of the plot. Calculation of Shape Difference Parameter is not changed if logarithmic scale is used.

In case of tests where the coverage of some regions of phase space is enhanced because of resonances, distribution properties are best visualized by distributions of Lorentz invariant masses constructed of all possible sub-groups of final state momenta. This is the key concept of the MC-TESTER methodology.

That was the case of τ lepton decays. Typically, one of the constructed invariant mass distributions was peaked around the position of the intermediate state resonance. The tails of the distributions are better populated when invariant masses are used directly, rather than higher powers of these invariant masses. To adopt for that type of applications, two options are

⁷Obtained respectively from the versions of PHOTOS where the matrix element is used (and where it is not).

introduced into MC-TESTER: the ability to plot given powers of the inv. mass, (see: A.1.2), and to scale the mass to its maximum possible value⁸ (see: A.1.3).

The histogramming of mass squared is useful if one is interested in eg. spin effects of $Z \rightarrow \tau^+\tau^-, \tau^\pm \rightarrow \pi^\pm\nu$ decays. The slope of the π^- energy in the Z rest-frame is proportional to the τ^- polarization, see fig. 2a. This spectrum is identical to the spectrum of invariant mass squared of the $\pi^+\pi^-\bar{\nu}$ system, and such distribution is now straightforward to study with MC-TESTER. For convenience we normalize the spectrum in proportion to its maximum possible value, that

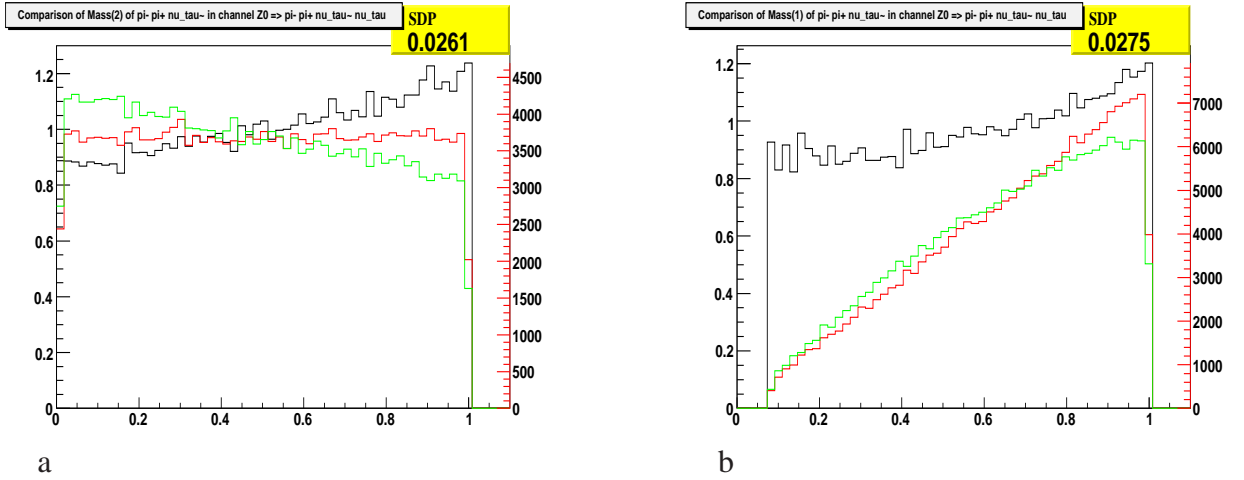


Figure 2: *Example of interesting benchmark for physics. In the decay $Z \rightarrow \tau^+\tau^- \rightarrow \pi^+\pi^-\nu\bar{\nu}$, the π^- energy spectrum in the Z rest-frame has identical shape as the distribution of the invariant mass squared of $\pi^+\pi^-\bar{\nu}$, see plot a. This linear distribution can be used to measure decaying τ^- polarization. The invariant mass is scaled to its kinematically allowed maximum. This is convenient when the mass of the decaying object is not constant. In plot b, the mass square option of MC-TESTER parameters is not used. In principle such a test is equally good to validate spin effects, but any discrepancies are more difficult to understand.*

is the invariant mass of the system of all (Z) decay products.

Without using those new options, physically equivalent plots could still be obtained, just plotting invariant mass, see fig. 2b. It is however far more difficult to interpret differences observed between compared Monte Carlo generators.

⁸The actually used power of the mass appears in the plot name as `Mass(2)` or `Mass(1)` respectively if mass square or mass itself is histogrammed.

4 Package organization (update for ref. [1])

This section contains technical details concerning MC-TESTER and should be used (together with reference [1]) as a quick guide book. Further details may be found in the Appendix and in files placed in the `doc/` subdirectory.

4.1 Directory tree

doc/ - contains documentation.

examples-F77/ - includes example programs in F77:

tauola/ - using the TAUOLA generator;

pythia/ - using the PYTHIA/JetSet generator;

pre-generated/ - results of generation with high statistics.

examples-C++/ - examples in C++:

pythia/ - example for the PYTHIA 8.1 generator;

evtgenlhc/ - example for the EvtGenLHC generator (CERN GENSER version).

analyze/ - analysis step is performed in this directory, the analysis code is contained in a set of ROOT macros.

prod1/ - (`mc-tester.root`) data file obtained from the generation phase with the first generator should be put here,

prod2/ - (`mc-tester.root`) data file from the results obtained with the second generator should be put here,

booklet/ - is created during the analysis step. It contains the result histograms in the form of `.eps` files.

HEPEvent/ - includes universal C++ interface to F77 event records (*i.e.* HEPEVT,LUJETS,PYJETS).

HepMCEvent/ - source code for `libHepMCEvent` (MC-TESTER interface libraries to the HepMC event record);

include/ - links to C++ include files.

lib/ - contains compiled libraries needed by MC-TESTER. Both the static and dynamic libraries are provided.

src/ - contains the source code for MC-TESTER.

platform/ - platform-dependent support files; currently only for Linux.

4.2 Libraries

The MC-TESTER source code is build into three libraries: `libMCTester`, `libHEPEvent` and `libHepMCEvent`. These libraries may be found in the `lib/` directory after installation.

The library `libMCTester` contains all the code needed by the generation step; it is also required at the analysis step, because it contains routines for the calculation of the Shape Difference Parameter.

The `libHEPEvent` library contains a unified interface for various F77 HEP Monte Carlo event record standards. In the current version of MC-TESTER, it provides unified access to the HEPEVT, LUJETS and PYJETS standards, enabling MC-TESTER to be used with a variety of Monte Carlo event generators, based on those event record standards.

MC-TESTER was recently extended to include the `libHepMCEvent` library. This library extends MC-TESTER's event record interface defined in `libHEPEvent` by adding support for the HepMC (versions greater than 2.0).

The source code of `libMCTester` is placed in the `src/` directory; `libHEPEvent` is stored in the `HEPEvent/` directory; and `libHepMCEvent` is stored in the `HepMCEvent/` directory.

4.3 Format and syntax of the SETUP.C file

The `SETUP.C` file is a C++ ROOT macro file, which controls MC-TESTER's settings. It is read and executed during initialization of both phases of a MC-TESTER run: the generation and the analysis. Nothing, except points discussed in the Appendix, require further explanation here. Documentation for the first version of the program is up to date.

4.4 How to make MC-TESTER run with other generators

4.4.1 The case of C++

The infrastructure for connecting MC-TESTER to a C++ generator was in place in early versions of MC-TESTER, but has recently been extended to allow analysis of C++ event records in HepMC v2.0 and greater. As in the case for F77 code, the generator program should be linked to the MC-TESTER libraries: `libMCTester` and `libHEPEvent`, as well as a subset of ROOT libraries. Configuration of MC-TESTER can be done either through a `SETUP.C` file (see Section 4.3) or directly in the body of the generation code (see Appendix A.2).

- **C++ generation program with FORTRAN event record**

All event record standards included in the current version of the `HEPEvent` library, i.e. HEPEVT, LUJETS and PYJETS may directly be used in a user's C++ code. It is sufficient to issue calls to the following three functions inside the tested Monte Carlo analysis.

1. `MC_Initialize()`: initializes MC-TESTER. All changes to the Setup should be commenced before a call to this function is invoked.
2. `MC_Analyze()`: performs the analysis of the event record specified in the `Setup::EVENT` variable;

3. `MC_Finalize()`: writes the results to the output file.

`Generate.h` contains definitions for the functions listed above, and should be included in the generation program.

An example of running MC-TESTER with `EvtGenLHC` (C++ with HEPEVT event record standard) can be found in the directory `examples-C++/evtgenlhc/`.

- **C++ generator with HepMC event record**

To analyze HepMC⁹ event records the additional interface library `libHepMCEvent` should be linked to the generation code, and `HepMCEvent.H` as well as `Generate.h` needs to be included. Inside the main (and to be tested) event generation program, MC-TESTER can be called in the following way:

1. `MC_Initialize()`: initializes MC-TESTER.
2. `HepMCEvent temp_event(HepMC::GenEvent event)`: creates an MC-TESTER interface event to the `HepMC::GenEvent` type event;
3. `MC_Analyze(&temp_event)`: performs an analysis of the `HepMC::GenEvent` passed to `temp_event`. The variable `Setup::EVENT` in `SETUP.C` will be ignored;
4. `MC_Finalize()`: writes the results to the output file.

An example of MC-TESTER's application to HepMC events can be found for PYTHIA 8.1 in the directory `examples-C++/pythia/`.

Each Monte-Carlo generators could potentially have its own set of particle status codes to record the types of processes a particle can undergo. For example in PYTHIA 8.1, negative integers are used for decaying particles, while in HERWIG positive integers are used. Therefore, it is important to document the way in which MC-TESTER interprets HepMC status code information to conclude if a particle is stable, decayed, or history/documentation (consistent with the HEPEVT standard codes of 1,2,3 respectively). The following definitions have worked successful with a variety of C++ MC Programs including PYTHIA 8.1 and Herwig++.

- **Stable:** If a particle has status code 1 or no end vertex of type `HepMC::GenVertex` it will be treated as stable by MC-TESTER. The particle will be placed in the list of daughters (see Section 5.) and the decay chain will not be traversed beyond this particle.
- **History:** If a particle has status code 3 it will be treated as documentation and MC-TESTER will not place it in the list of daughters. The decay chain will not be traversed beyond this particle.
- **Decayed:** All other particles will be classified as decayed. The treatment of these particles by MC-TESTER depends on the configuration of `SETUP.C` (see also A.2).

⁹Only HepMC version 2.0 or greater is supported.

For the use of status codes beyond the above definitions, MC-TESTER must be run with a modified version of the event record (see Section 4.5).

Filtering of the daughter list can be achieved using `UserTreeAnalysis` (see Appendix A.1.1).

4.5 How to make MC-TESTER run with on-flight modified event records

Often we want to pass into MC-TESTER, not the exact event tree as stored in the event record, but a modified one. To date, `UserTreeAnalysis` (see A.1.1) has been used for this purpose. It provides a solution which works quite well, and hence will be discussed in the next section. Numerous options can be introduced in this way. However, methods are limited to manipulation during creation of the list of stable (endpoint) objects, originating from the analyzed object, or later. Some analysis of the object itself can be introduced as well. The access to its origin or eg. kinematical properties may be interesting.

This requires that the event record logically matches our requirements. This is not necessarily always the case and one is tempted to perform some adjustment. In short, one may be interested in HepMC to HepMC translation.

We have realized that such modules can be of interest for other applications independent from MC-TESTER as well. We will not discuss the related topic here at all. For practical use one need to call a method, which creates a temporary copy of the modified HepMC event accordingly.

The sequence of calls, which can serve as a possible template for future work, may look as follows:

```
HepMCaid HepMCnew;  
HepMCnew.SetOption(...);  
.....  
HepMCnew.ModifyEvent(HepMCold);  
MC_Analyze(HepMCnew);
```

5 The use of lists in MC-TESTER

MC-TESTER was developed over many years of experience in extracting data from different variants of event records. This section addresses the readers who are interested in this aspect of activity. Our past experience may be of future use, it may explain some of the reasons behind our program design too.

Even though the idea of a standardized event record structures has been in place for a long time, technical problems arose. For programs coded in FORTRAN, event records (such as HEPEVT or LUJETS/PYJETS) were typically expressed as integer-indexed arrays, with every index representing a single item, such as a particle. To express relationships between the particles or the ordering, one had to use the arrays of indices, as FORTRAN did not provide for more complex data structures such as lists. The emulation of such structures through the arrays of indices used in the standards such as HEPEVT soon reached its limits: the arrays that were designed to store

the indexing information for doubly-linked lists representing the decay tree started to fail, due to information overload: the place designated for the storage of pointers in the emulated decay tree structure was re-used to store additional information. The - initially easy - way to navigate through the tree-like structure of the decay cascade became inconsistent, generator-dependent and non-trivial to interpret. Moreover, to compensate for the oversimplified model of the event structure, the authors of the event generators started to introduce their own conventions, to fit additional information about the intermediate, internal states of the particles and objects, which could not be expressed directly¹⁰.

Energy-momentum conservation, at the tree branchings, was not always assumed; branching points had to be ‘understood’ as part of the larger group. The original arrays that stored mother and daughter pointers to encode the decay process started to serve multiple purposes, depending on the context: either (as originally) to indicate the decay processes, or to encode some other relationship. Needless to say that for the programs such as PHOTOS [7–10], which relies on consistent information about the decay cascade structure to extract sub-trees, or TAUOLA [5, 6], which needs to extract the information about the hard process, it has become extremely difficult to maintain a consistent generic and reliable interface to other event generators.

The same type of problems obviously manifested in the large detector-simulation chains, where the input from a theory-rich physics Monte Carlo event generator needed to be combined with the phenomenological description of the detector processes, to give observables such as energy-deposits and hits, which are in turn re-processed to reconstruct the original process. Extracting the “signal” process from an event record structure with overloaded data, full of generator-specific conventions made this task, again, difficult and prone to errors; additional analysis steps (which merely had to compensate for the insufficient data model of the event record) had to be fitted to enable the use of each new event generator.

In the earliest version of MC-TESTER [1] we suffered from the same type of problems with index-based navigation through the event records. However, since version 1.1, we decided to take advantage of possibilities given by the C++ programming language, and employ more complex data structures in the processing performed by MC-TESTER. The `libHEPEvent` interface was extended with the abstraction of “particle list”. The particle list (`HEPParticleList` class) could store pointers to any number of objects representing the particles in the event record. A set of methods, modelled on the concept of iterators from the C++ Standard Template Library was added as well, to facilitate the navigation in the list object, effectively replacing the native constructs of index-based “FOR” or “DO” loops. The previously used methods to navigate through the event tree using the indices were replaced with the list: each type of event record (supported by `libHEPEvent`) had a new method that returned a list of child particles, and the associated iterator object was implemented in such a way that all the generator-specific conventions were hidden in it, presenting a clean and simple-to-use interface. The code of MC-TESTER was modified to make use of these new constructs - it gained significantly in clarity and stabil-

¹⁰As a result, one could see in the encoded decay trees, cases such as a τ particle decaying to another τ and a photon, and then this second τ decaying to yet some other particles with both of the τ 's being actually two instances of the same particle, and the fact of having them listed twice in the event record was to express their state at two distinctive stages of event construction, and express the bremsstrahlung processes of τ production rather than decay.

ity: not only the previously-used, index-based syntax being replaced with the constructs native to Object-Oriented languages, but also the possible dependencies (or incompatibilities) could be delegated to be served within another module (`libHepEvent`), making the code much easier to maintain. Ultimately, this also enabled the implementation of the `HepMCEvent` interface, and the use of `HepMC`-based event generators with `MC-TESTER`, in a straightforward way.

The mechanism that extracts the lists of "daughter" particles, which is currently implemented in the `libHEPEvent`, still does not address the more fundamental problem of oversimplification in the so-far proposed event record structures, including `HepMC`: the inability to express other types of relations between particles necessary for more complex models of processes where for example quantum interference need to be included. Up to now, there is only one type of "relations", being mother-daughter relations in the old event records, or "interaction vertices" or "blobs" in the newer ones: they only express the relation in the decay cascade. Other types of relations or processes still need to be "emulated" by employing special conventions, or additional, often non-physical objects (many instances of the same particle, etc). As already discussed, the resulting data structure is difficult to interpret: objects as physical entities, the actual processes taking place during (often multi-step) event generation, and to implement "content enriching" generators, such as `TAUOLA` and `PHOTOS`, which add to an already (partially) generated event, stored in the event record. Up to this point, we treat the exercises with list-based methods for "re-interpretation" of the event-record data, as an initial seed for a more concentrated effort to provide such "re-interpreting" code in the near future, targeted in particular as helper utilities for the software of large experimental collaborations. Additional ingredients for such utilities are provided by the experience we gained with the "user tree analysis" feature of `MC-TESTER`, documented in Appendix A.1.1 and Section 4.5, where we extract/re-construct/correct a fragment of the event record "on the flight", and present it as the input to `MC-TESTER`, rather than using the original event record. Such an approach, with re-engineered, re-interpreted event data, created on-the-flight, using a simple-to-use, pluggable script/macro files has an additional advantage: the original event records remains unmodified, and could still be accessed.

`MC-TESTER` is not the only of our projects, where the problems discussed above have to be addressed. `TAUOLA universal interface` [6] and `PHOTOS` [8] represent further examples. Our program is devoted to tests, this is why it was worked out before the other two.

Similar solutions to problems as those discussed here [15], are possible. It seems that in this respect, the case of `TAUOLA universal interface` is easier than `PHOTOS`. New objects need to be added to the end points of the otherwise unmodified tree. The prototype solution, based on `HepMC` exist already [16].

The standard concept of C++ lists can be used for minor practical adaptation in `MC-TESTER` too. For example, at the time of list creation one can force some particles to be treated as stable, and its consecutive daughters ignored.

6 Example of advanced MC-TESTER use: benchmarks for spin correlations in heavy object decays.

One may have the impression that the modifications introduced into the present release of the package are minor and consist of simple improvements in the graphical representation of the output and purely technical reorganization thanks to the use of C++ lists.

To some degree this is true, but other changes were introduced because of pressure from applications. In the present chapter, let us show, how program modifications can be used for non-trivial practical applications.

It is quite common that information stored in the event record is too large. For example individual soft photons which remain undetectable are present. Not only they do not influence the detector response at all, but they exhibit technical aspects of eg. infrared regulators of QED bremsstrahlung. In response, MC-TESTER should ignore (or group together with other particles), those photons while analyzing decays. Otherwise comparisons of different Monte Carlo programs would be dominated by the technical aspect of the implementation of infrared regulator; MC-TESTER operation need to be adopted for this, see eg. [9, 10].

Another example where the event tree may need to be simplified for validation is if spin correlations are appropriately introduced into various production processes. Let us use as an example¹¹ $pp \rightarrow Z/\gamma^* + X, Z \rightarrow \tau^+\tau^-$. It is convenient to start the test by restricting τ decays to the simplest decay mode, that is $\tau^\pm \rightarrow \pi^\pm \nu$, and look at distributions in combined decay $Z \rightarrow \pi^+\pi^-\nu\bar{\nu}$. In this case the effects of spin correlations are largest. The distribution of the π^- energy spectrum (in the Z rest-frame), manifests the τ polarization through its slope (see fig 2a). Fortunately, this frame dependent spectrum is equal to the distribution of the invariant mass squared of $\pi^-\pi^+\bar{\nu}$. This distribution can be obtained in MC-TESTER thanks to the new options of histogramming invariants squared (A.1.2) and automatic scaling of the histogram range to the kinematically allowed maximum (A.1.3).

Final state activities will lead to Z decays where the τ -pair is accompanied eg. by bremsstrahlung photons or soft hadrons. One may want to ignore this soft radiation in the test, or quite contrary – look only at these cases, to verify if soft emissions did not result from configurations of faulty spin correlations. Finally one may want to check decays of Z of high p_T only.

On the other hand all such variants of non-standard MC-TESTER analysis were rather easy to include into our example of the `UserTreeAnalysis` (see. A.1.1), but we assume that in the future other options may also become useful. On the other hand some interesting variants of the `UserTreeAnalysis` method may not be possible using decay product lists alone. For example if one would be interested in decays of Z s originating from an object X or accompanied in X decay by eg. another Z or top quark. For that purpose some other methods following the idea of `userEventAnalysis` [1] or `HepMCaid` (see 4.5) may be useful.

6.1 Default `UserTreeAnalysis`

`UserTreeAnalysis`, is included in the source code of MC-TESTER thus can be loaded with the

¹¹Spin correlations in decays of W, H, H^\pm into τ lepton(s) are nearly identical

library `libMCTester`. The user can create his own version of the method named eg. `MyUserTree`¹², and load it as a pre-compiled C++ macro instead. The parameters of the built-in `UserTreeAnalysis` named "`UserTreeAnalysis`", have the following meaning:

1. `params[0]=0.05` minimum value of the variable used to discriminate soft particles as a fraction of the decaying particle mass.
2. `params[1]=0` maximum number of possible soft particles retained, even if passing a threshold of the previous option.
3. `params[2]=0` type of variable used in discrimination
 - (a) 0 - energy in the decaying particle rest frame
 - (b) 1 - energy in lab frame
 - (c) 2 - p_T in lab frame
4. `params[3]=0`
 - (a) 0 - removed particles are simply ignored
 - (b) 1 - removed particle momenta are added to the momenta of charged ones. For details see [1].
5. `params[4]=22` PDG Id's of the particles to be removed. Repetition of this parameter is allowed for `params[5]`, `params[6]` etc.

If parameters are not initialized, the default values, as given in the points above, are used. For the example, our method defined three histograms for the properties of the decaying particle: its p_T , pseudorapidity and azimuthal angle ϕ . They are included, during the analysis step, in the "User Histogram" section of the MC-TESTER booklet.

This simple method summarizes and extends the technical aspects of tests we have developed in papers [9, 10].

7 Outlook

We have demonstrated that MC-TESTER may be useful for tests of libraries of particles decays, as well as for tests of their interfaces (see fig. 2 and refs. [5, 16] for example).

The updated version of the package was found [9, 10] to handle well cases where physically spurious information (eg. on soft photons) need to be ignored. This avoided unphysical discrepancies between results from different programs. Moreover, adaptations of the program may lead to a new spectrum of applications, which, as discussed in Sections 4.5, 5 and 6, may find applications independent of the future evolution of the MC-TESTER software project.

¹²See Appendix A.1.1 for details.

Even with the present enrichment of functionality, the tests performed by MC-TESTER are not complete from the physics point of view. The program has also some technical limitations. In the following we list these points, which may be addressed in future versions of MC-TESTER and require stating.

1. The program does not analyze distributions in Lorentz invariants built with the help of the totally antisymmetric (Levi-Civita) tensor. It is thus blind to some effects of parity non-conservation.
2. Information on the spin state of the decaying particle is usually not available in event record structures such as HepMC. To keep MC-TESTER modular, and to avoid a multitude of options, we ignore effects of decaying particle polarization.
3. The main advantage of MC-TESTER is that it can be used with ‘any’ production generator in an automated way, providing a tool for quick tests. However, the final state event record has to be stored in one of the following structures: common blocks HEPEVT, LUJETS, PYJETS [17, 18] of FORTRAN or HepMC objects of C++.
4. If multiplicity of the particular decay channel is very high and/or there is a lot of decay channels, the program may find it difficult to allocate memory. An analysis of a decay channel with 8 or more decay products results in thousands of histograms, which causes output files to be large and the analysis step to be long. Hard coded limits have been implemented: Histograms will only be created for the first 200 decay channels found and only if the multiplicity of decay products is smaller than 8.
5. Some of MC-TESTER’s options, especially MyUserTree method, see Section 6.1, may be difficult to use within large systems like Athena¹³ of the ATLAS collaboration. This point will also need to be investigated after the release of the present version of our program. It requires interaction with the users.

The main purpose of MC-TESTER is to analyze sub-trees starting from the objects of a given PDG identifier without any concern of its origin, and without pre-selecting the type of distributions created. This is why there is complementarity between our approach and the one of Rivet [19]. The latter is designed to produce simulated distributions which can be directly compared to measured data for validation and tuning purposes. The MC-TESTER strategy is to test decays on a technical level at the event record content first, rather than to start from already pre-identified quantities of physics interest. Only later one may, but with constrained possibilities only, turn to physically interesting quantities.

Updates introduced to the program after version 1.23 become public are described in Appendix B.

¹³<http://atlas.physics.utoronto.ca/Members/bguo/setup-athena-12-0-0-at-cern-machines>

Acknowledgments

We thank Elzbieta Richter-Was, Alberto Ribon, Judith Krantz and Zhonghua Qin for comments on the program organization and documentation. Nadia Davidson would like to thank the “Marie Curie Programme” for her fellowship. Partial support of Polish-French collaboration no. 06-124 within IN2P3 through LAPP Annecy during final completion of this work is also acknowledged.

A Appendix: MC-TESTER setup and input parameters (update for ref. [1])

The values of the parameters used by MC-TESTER are controlled using the `SETUP.C` file. Some parameters may also be controlled using FORTRAN77 interface routines or C++ methods (Section A.2). This provides runtime control over all parameters, yet allowing the user not to have `SETUP.C` at all. One should note that `SETUP.C` always has precedence over the default values set using F77 or C++ code: it is always looked for in the execution directory.

A.1 Definition of parameters in the `SETUP.C` file

There are three sets of settings inside MC-TESTER to be distinguished: the ones specific to the generation phase, the ones specific to the analysis phase and the ones that are used in both phases¹⁴. We describe only new features, quoting the scope of their use.

A.1.1 Setup::UserTreeAnalysis

Type: `char*`

Scope: generation

Default: null

DESCRIPTION: The name of a function that allows modification of the list of stable particles before histograms for the decay of MC-TESTER analyzed object are defined/filled in.

IMPORTANT: The name that is attributed (eg. "MyUserTree") must be a valid method name existing in a C++ script file located in the working directory. The script must be given the same name as the method, and ended with a ".C" suffix. For example, for the "MyUserTree" method, the script filename would be `MyUserTree.C`. For further information on running and compiling scripts on the fly see `README.UserTreeAnalysis` in the `MC-TESTER/doc/` directory.

Example of use:

```
Setup::UserTreeAnalysis = "MyUserTree";
```

or for the version compiled and present in `libMCTest` library:

```
Setup::UserTreeAnalysis = "UserTreeAnalysis";
```

In this case, the `UserTreeAnalysis.C` is not needed, as the built-in `UserTreeAnalysis` routine will be used.

Parameters can be passed to the function. For example

```
Setup::UTA_params[0]=0.05;
```

```
Setup::UTA_params[1]=0;
```

```
Setup::UTA_params[2]=0;
```

```
Setup::UTA_params[3]=0;
```

```
Setup::UTA_params[4]=22;
```

```
Setup::UTA_params[5]=111;
```

¹⁴Some parameters from the generation phase (i.e. the description of generators) are stored inside an output data file. However, again for reasons of runtime control, their values may be altered at the analysis time using the `SETUP.C` file in the analysis directory.

`Setup::UTA_nparams=6;`
will pass `nparams=6` parameters to the function. For the actual meaning of the parameters if passed into `UserTreeAnalysis` as present in the library, see section 6.

A.1.2 `Setup::mass_power`

Type: int

Scope: generation

Default: 1

DESCRIPTION: This option changes the variable passed for histogramming, from invariant mass to a power of invariant mass, at the generation step. It also modifies the title displayed on histograms from `Mass(1)` to an appropriate `Mass(value)`, showing that the power of the mass has been changed.

NOTE: Acceptable values: from 1 to 9. Due to properties of the Lorentz group when this option has `value=2` it is particularly suitable for tests of spin polarization, see section 6.1.

Example of use:

```
Setup::mass_power=2; //set histograms to invariant mass squared
```

A.1.3 `Setup::mass_scale_on`

Type: bool

Scope: generation

Default: false

DESCRIPTION: This option scales invariant masses for all plots of the decay channel to invariant mass constructed from all daughters combined. It scales the X values to the range (0,1).

NOTE: When using this option consider setting default maximum bin value to 1.1, for nicer graphical representation.

Example of use:

```
Setup::mass_scale_on=true; //enables scaling of X axis
```

A.1.4 `Setup::use_log_y`

Type: bool

Scope: analysis

Default: false

DESCRIPTION: Enables the use of logarithmic scale in all histograms plotted by `MC-TESTER`. Turning this option on will draw the histograms in logarithmic scale, and mark a logarithmic scale along the right-hand-side Y axis. This option does not affect SDP calculation or the plot of the ratio of histograms, which remains linear. Its corresponding linear scale is marked on the left-hand Y axis.

NOTE: This option, combined with previously presented defaults for `UserTreeAnalysis` can be particularly useful if infrared regulator sensitive particles, such as soft photons are present in the event records. See [20].

Example of use:

```
Setup::use_log_y=true; //enables logarithmic scale on Y axis
```

A.1.5 Setup::rebin_factor

Type: int

Scope: analysis

Default: 1

DESCRIPTION: One may want to define a large number of bins for the generation scope of MC-TESTER. The number of bins on the actual plots, can be adjusted at the analysis step. The contents of consecutive "rebin_factor" bins are summed together. Calculation of the SDP parameter is appropriately adjusted.

NOTE: "rebin_factor" must be the natural divider of the number of bins declared during the generation scope of MC-TESTER.

Example of use:

```
Setup::rebin_factor=3; //reduces no. of bins in all histograms by factor of 3
```

A.2 C++ configuration of MC-TESTER

The configuration of MC-TESTER can be done directly in the main method of the C++ generation program, without the need for a SETUP.C file. This can be accomplished by including the header file Setup.H and setting parameters using the same syntax as described for SETUP.C files (see original documentation [1]). Setup should be done before calling the function MC_Initialize(). Note that if parameters are set in both the generation program and a SETUP.C file, the values present in SETUP.C will be given precedence.

B Appendix: updates from version 1.23 up to version 1.24.4

B.1 Changes introduced in version 1.24.2

To address the problems that are typically faced when MC-TESTER is installed in a new environment, or a new platform, an automated configuration step has been implemented in version 1.24.2. The configuration files required to set-up/compile/run MC-TESTER may be generated through a dedicated configuration script, which facilitates the GNU autoconf [21].

To set up MC-TESTER using the new auto-configuration facility, proceed with the following steps:

- Execute ./configure with additional command line options:
 - with-HepMC=<path> provides the path to HepMC installation directory (alternatively HEPMCLOCATION system variable has to be set).
 - with-root=<path> Path to root binaries.
 - with-Pythia8=<path> Path to Pythia version 8.1 or later (this generator is used by

examples only)

`--prefix=<path>` provides the installation path. If this option is used `include/` and `lib/` directories will be copied to this prefix `<path>` when `make install` will be executed. If `--prefix=<path>` is not provided, the default installation directory `/usr/local` will be used

- Execute `make`; this will build `MC-TESTER`.
- To install `MC-TESTER` into the directory specified at step 1) through the `--prefix` parameter, execute `make install`; this will copy the include files and libraries into `include/` and `lib/` sub-directories.
- It is worth to mention that `./configure` scripts only prepare `make.inc` files. These files are rather short and can be easily modified or created by hand: one can also rename `README-NO-CONFIG.txt` to `make.inc` and modify it accordingly to instructions provided inside the file.

Further changes and bug-fixes were implemented too. However they do not require any changes in the way the program is used. Let us nonetheless list them here:

- A bug resulting in faulty functioning of the script `ANALYZE.C` was fixed. Previously, when comparing decay samples which differed by several distinct channels, the program was occasionally crashing.
- A bug resulting in faulty functioning of `UserTreeAnalysis` scripts was fixed. The program was crashing if `MC4Vector` was used inside the script.
- All offending statements resulting in compilation errors if `'-ansi -pedantic'` flags were activated have been removed now.

B.2 LCG configuration scripts; available from version 1.24.2

For our project still another configuration/automake system was prepared by Dmitri Konstantinov and Oleg Zenin; members of the LCG/Genser project¹⁵ [22, 23].

For the purpose of activation of this set of autotools-based installation scripts enter `platform` directory and execute there `use-LCG-config.sh` script. Then, installation procedure and the names of the configuration script parameters will differ from the one described in our paper. Instruction given in `'./INSTALL'` readme file created by `use-LCG-config.sh` script should be followed. One can also execute `./configure --help`, it will list all options available for the configuration script.

A short information on these scripts can be found in `README` of main directory as well.

¹⁵We have used the expertise and advice of that team members in organization of configuration scripts for our whole distribution tar-ball as well.

B.3 Merging MC-TESTER output files; available from version 1.24.3

Interest in using the program on distributed systems, such as the grid has been expressed on several occasions. This calls for new functionality: to merge several `mc-tester.root` files into a single one, corresponding to all event samples combined into one.

The `analyze/MERGE.C` script can be used for this purpose:

- Enter the `analyze/` directory.
- Execute `root -b MERGE.C` (or `root -b and .L MERGE.C`).
- Type `merge(<output file> , <input directory>/<first file> , [<pattern>])` (the last parameter is optional).
- Copy `<output file>` into `analyze/prod1/mc-tester.root` (or `analyze/prod2/mc-tester.root`).

Example:

```
root -b
root [0] .L <path_to_script>/MERGE.C
root [1] merge("out.root", "samples/first.root", "*.root")
```

The input to the script may consist of just the `<input directory>` path where reside the `.root` files to be merged. Alternatively, the name of the first MC-TESTER `.root` file to be merged (`<input directory>/<first file>`) can be explicitly given. Generator information will be taken from this first file. The script will search the `<input directory>` to merge all files matching the pattern. If no pattern is provided, the default pattern is `mc-tester*.root`. The `<output file>` will feature all histograms for decay channels including user defined histograms. Histograms for decay channels of the same name, found in different files will be summed together. The histogram bin count and axis range of the first occurrence will be used.

If histograms are found with a distinct axis range or number of bins, compared to other histograms with the same name, then the content of these files is ignored. However, if by mistake, the particular input file contains data for tests of another particle's decays, then all data from this file will be taken. All decay channels for all particles under consideration will be listed in the `.pdf` file constructed by MC-TESTER at the analysis step. Information printed on the front page might then be inconsistent, for example, the overall number of entries or the overall number of channels will represent decays of all particles. If the interest will be expressed in future, an analysis step can be adopted to handle such cases with better front page of the booklet.

If the script is used outside the `MC-TESTER/analyze/` directory, the `MCTESTERLOCATION` system variable needs to be set to the `MC-TESTER` root directory. In addition, user can create and adopt his own copy of `MERGE.C` and use it instead of the default one. Note that our script cannot be used with a version of MC-TESTER older than 1.24.3.

B.4 Updates introduced in version 1.24.4

Version 1.24.4 differs from 1.24.3 by comments introduced into all parts of the code. Thanks to their introduction also Doxygen documentation is sufficiently exhaustive. Bug resulting in a faulty printout of histogram names was also fixed. Finally modification of `MERGE.C` script, necessary for proper handling of User Histograms, was introduced.

References

- [1] P. Golonka, T. Pierzchala, and Z. Was, *Comput. Phys. Commun.* **157** (2004) 39–62, hep-ph/0210252.
- [2] S. Jadach, Z. Wąs, R. Decker, and J. H. Kühn, *Comput. Phys. Commun.* **76** (1993) 361.
- [3] M. Jezabek, Z. Wąs, S. Jadach, and J. H. Kühn, *Comput. Phys. Commun.* **70** (1992) 69.
- [4] S. Jadach, J. H. Kühn, and Z. Wąs, *Comput. Phys. Commun.* **64** (1990) 275.
- [5] T. Pierzchała, E. Richter-Wąs, Z. Wąs, and M. Worek, *Acta Phys. Polon.* **B32** (2001) 1277–1296, hep-ph/0101311.
- [6] P. Golonka *et al.*, *Comput. Phys. Commun.* **174** (2006) 818–835, hep-ph/0312240.
- [7] E. Barberio, B. van Eijk, and Z. Wąs, *Comput. Phys. Commun.* **66** (1991) 115.
- [8] E. Barberio and Z. Wąs, *Comput. Phys. Commun.* **79** (1994) 291–308.
- [9] P. Golonka and Z. Was, *Eur. Phys. J.* **C45** (2006) 97–107, hep-ph/0506026.
- [10] P. Golonka and Z. Was, *Eur. Phys. J.* **C50** (2007) 53–62, hep-ph/0604232.
- [11] <http://root.cern.ch/root/Availability.html> .
- [12] T. Sjostrand, S. Mrenna, and P. Skands, *Comput. Phys. Commun.* **178** (2008) 852–867, 0710.3820.
- [13] M. Dobbs and J. B. Hansen, *Comput. Phys. Commun.* **134** (2001) 41–46, <https://savannah.cern.ch/projects/hepmc/>.
- [14] D. Lange and A. Ryd, <http://lhcb-release-area.web.cern.ch/LHCB-release-area/DOC/gauss/generator/evtgen.php> and <http://www.slac.stanford.edu/lange/EvtGen/>.
- [15] Z. Was, Prepared for Workshop on Computer Particle Physics: (CPP 2001): Automatic Calculation for Future Colliders, Tokyo, Japan, 28-30 Nov 2001.
- [16] N. Davidson, G. Nanava, T. Przedzinski, E. Richter-Was, and Z. Was, 1002.0543.
- [17] Particle Data Group Collaboration, C. Caso *et al.*, *Eur. Phys. J.* **C3** (1998) 1.
- [18] T. Sjostrand *et al.*, *Comput. Phys. Commun.* **135** (2001) 238.
- [19] A. Buckley *et al.*, <http://projects.hepforge.org/rivet/>
<http://projects.hepforge.org/rivet/trac/wiki>.
- [20] P. Golonka, G. Nanava, and Z. Was, Tests of PHOTOS Hard Bremsstrahlung, <http://mc-tester.web.cern.ch/MC-TESTER/PHOTOS-MCTESTER/>.

[21] GNU Autoconf <http://www.gnu.org/software/autoconf/>.

[22] LCG project <http://lcg.web.cern.ch/LCG/>.

[23] M. Kirsanov, A. Ribon, and O. Zenin, *PoS ACAT08* (2008) 114, See also: <http://lcgapp.cern.ch/project/simu/generator/>.