CERN-PH-TH/2010-261
IFJPAN-IV-2010-6
November 2010

PHOTOS Interface in C++

Technical and Physics Documentation

N. Davidson[a,b], T. Przedzinski[c], Z. Was[a,d]

[a] *Institute of Nuclear Physics, Polish Academy of Sciences, ul. Radzikowskiego 152, 31-342 Cracow, Poland*

[b] *The University of Melbourne, School of Physics Australia*

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

[d] *Theory Group, Physics Department, CERN, CH-1211, Geneva 23, Switzerland*

Abstract

The first version of PHOTOS Monte Carlo for bremsstrahlung in the decay of particles and resonances with an interface to the HepMC event record written in C++ is now available. The main purpose of the present paper is to document technical aspects of the PHOTOS Monte Carlo installation and its use. A multitude of test results and examples are distributed together with the program code.

The PHOTOS C++ physics precision is now as good as that of its FORTRAN predecessor. However better steering options are available. An algorithm of the event record interface is prepared for the installation of process dependent variants of the photon emission kernel. Weights, featuring complete first order matrix elements, can be installed for general use. In the FORTRAN version of PHOTOS they were available only for decays of particles at rest and with spin set along the z axis.

Physics assumptions used in the program and properties of the solution it offers are reviewed. In particular, it is mentioned that the second order matrix elements were used in design and validation of the program iteration procedure. Also it is explained that the phase space parameterization used in the program is exact.

† 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 (2008-2011).

Table of Contents

1 Introduction

For a long time, PHOTOS Monte Carlo [?, ?] has been used for the generation of bremsstrahlung in the decay of particles and resonances. Over the years the program has acquired popularity and it evolved into a high precision tool [?]. Since version 2.15 of the program become public in the year 2005 and multi-photon radiation was introduced [?], there were no further public upgrades of the program. However the effort on documentation and tests was going on; phase space treatment was shown to be exact [?] and for several processes [?, ?, ?] an exact matrix element was studied with the help of optional weights. Benchmark distributions, including comparisons with other simulation programs, were collected on the program web page [?].

Such high precision applications require good control of the event record content on which PHOTOS operates. On one side it requests skills and experience of the user and on the other it provides the flexibility necessary for the study of effects like, for example, systematic errors for measurements of anomalous couplings or W cross section. Methods of correlated samples can be applied[1].

Until recently HEPEVT [?] was used as the structure for communication between physics Monte Carlo programs and detector/reconstruction packages. Experimental physicists used HEPEVT for their own applications as well. Recently, to gain flexibility, FORTRAN is being replaced by C++ and instead of HEPEVT, the C++ event structure called HepMC [?] is used. Nothing prevents moving PHOTOS to an environment based on HepMC and to rewriting the whole (beginning with the event record interface) of PHOTOS[2] to C++. In fact implementation of the algorithm in that language is clearer and easier to maintain. Because of its design the PHOTOS algorithm benefits from the object oriented features of C++. It is our third program, after MC-TESTER [?] and the TAUOLA interface [?], previously ported to HepMC and C++. This completes the main step of migration of these three programs to the new style.

Such migration is convenient for the users too; they can now work with homogeneous C++ software. From the physics point of view, transformation of PHOTOS from FORTRAN to C++ brings some benefits as well. The channel dependent, complete first order matrix elements of PHOTOS, in FORTRAN, are available only in special kinematical configurations. With the help of the new event record interface they will become available for general use. For that purpose, better access to the information necessary to orient the spin state of decaying particles is now provided.

Our paper is organized as follows. Section 2 is devoted to a description of physics information which must be available in the event record for the algorithm to function. Later, particular requirements for the information stored in HepMC are given. Section 3 describes the program structure. In Section 4 methods prepared for future extensions to improve the physics precision of the generator are explained. Section 5 presents the program tests and benchmarks. A summary, section 6, closes the paper. There are three appendices A, B and C attached to the paper. Respectively, they describe the interface to the old FORTRAN part of the code, provide a user guide and explain the program configuration methods and parameters.

[1] To exploit such methods in the high precision regime, good control of matrix element properties is necessary. As was shown in [?], complications for such methods arise at the second order matrix element only, thus at the precision level of $(\frac{\alpha_{QED}}{\pi})^2 \simeq 10^{-5}$.

[2] An up-to-date version of the code described in this paper is available from the web page of our project [?].

2 Requirements of the PHOTOS Interface

The algorithm of PHOTOS Monte Carlo can be divided into two parts. The first, internal part, operates on elementary decays. Thanks to carefully studied properties of the QED (scalar QED) algorithm, with certain probability, it replaces the kinematical configuration of the Born level decay with a new one, where a bremsstrahlung photon or photons are added and other particle momenta are modified. This part of the program is sophisticated from the physics point of view [**?**, **?**], but from the point of view of data structures the algorithm is simple. That is why the gain from re-writing this part of the program to C++ is rather limited and will be postponed to a later step of the project development. On the other hand, there are not many obstacles for such a transformation to be performed. In fact it was already done previously [**?**], but the resulting program was developed too early and did not attract users because of a lack of a C++ event record format standard at that time.

The typical results of high energy process simulation are events of complex structure. They include, for example, initial state parton showers, hard scattering parts, hadronization and finally chains of cascade decays of resonances. A structure similar to a tree is created, but properties of such data structures are sometimes violated. For its action, PHOTOS needs to scan an event record (tree) and localize branchings where it is supposed to act. The decaying particle (mother) and its primary decay products (daughters) have to be passed into the internal event structure of PHOTOS. Finally for each daughter a list of all its subsequent decay products has to be formed. Kinematical modifications need to be performed on all descendants of the modified daughter.

In the new, C++ version of this part of the algorithm, additional functionality is added. The first mother of the decaying particle will be localized and passed together with the elementary branching to the internal part of the program. Prior to activation of the algorithm for photon(s) generation and kinematic construction, the whole decay branching (supplemented with its mother(s)) will be boosted into the decaying particle's rest frame and the first mother will be oriented along the z axis. In many cases, the spin state of the decaying particle can be calculated from kinematics of its production process. Later it can be passed on to the code which calculates the matrix element for our branching.

At present, the part of the code responsible for photon(s) generation and kinematic construction is left in FORTRAN. It does not feature the options presented above yet.

Before an actual description of the program, let us list the tasks the event record interface must be able to perform:

1. a method to read particles stored in the event tree.
2. a method to add or modify particles of the event tree.
3. a method to search for elementary decays over the entire tree of the event.
4. a method to form lists of all subsequent decay products originating from each elementary decay product.
5. a method to localize the first mother of the decaying particle.
6. a method to localize the second mother for a special case of $t\bar{t}$ pair.

2.1 C++ and HepMC Specific Requirements

The C++ version of the PHOTOS interface implements all functionality of its predecessor, the PHOTOS version 2.15 [?] coded in FORTRAN. The program is prepared for installation of the process-dependent correcting weights of refs [?, ?] into its public available version. PHOTOS can be attached to any Monte-Carlo program, provided that its output is available through a HepMC [?] event record. It seems that HepMC will remain a generally accepted standard for the near future. However, already now several different options for how HepMC is used are widespread. The possibility of the flexible adaptation of our event record interface to different options has been considered in the design, drawing experience from MC-TESTER [?, ?]. We have also envisaged the possibility that HepMC may one day be replaced by another standard of event record, and we have provided a way to extend the interface to a possible new event record standard as well.

2.2 Object Oriented Event Records – The Case of HepMC

In adapting the PHOTOS interface to the C++ event record format the difference between the HEPEVT event record, used in the FORTRAN version of the PHOTOS interface, and the HepMC event record, which is used for the C++ based interface, has to be taken into account. In the first case a FORTRAN common block containing a list of particles with their properties and with integer variables denoting pointers to their origins and descendants is used. The HepMC event structure is built from vertices, each of them having pointers to their origins and descendants. Links between vertices represent particles or fields. Fortunately, in both FORTRAN and C++ cases, the event is structured as a tree[3]; the necessary algorithms are analogous, but nonetheless different. The HepMC structure based on vertices is more convenient for the PHOTOS interface.

In HepMC, an event is represented by a GenEvent object, which contains information such as event id, units used for dimensional quantities in the event and the list of produced particles. The particles themselves are grouped into GenVertex objects allowing access to mother and daughter particles of a single decay. Vertices provide an easy way to point to the whole branch in a decay tree that needs to be accessed, modified or deleted if needed. The information of a particle itself is stored in a GenParticle object containing the particle id, status and momentum as well as information needed to locate its position in the decay tree. This approach allows traversing the event record structure in several different ways.

The HepMC event record format is evolving with time, making it necessary to adapt the code to new versions. The HepMC versions 2.06, 2.05 and 2.03 were used in the final tests of our distribution.

Evolution of the HepMC format itself is not a crucial problem. In contrast, conventions on how physics information is filled into HepMC represent the main source of technical and also physics challenges for our interface. This is quite similar to the previous HEPEVT - FORTRAN case. Let us discuss this point in more detail now.

[3]At least in principle, because in practice its properties may be rather like a graph without universally defined properties. This makes our task challenging.

2.2.1 Event Record Structure Scenarios

While many Monte-Carlo generators (e.g. `PYTHIA 8.1` [**?**], HERWIG++ [**?**]), SHERPA [**?**] can store events in `HepMC` format, the representations of these events are not subject to strict standards, which can therefore vary between Monte-Carlo generators or even physics processes. Some examples of these variations include the conventions of status codes, the way documentary information on the event is added, the direction of pointers at a vertex and the conservation (or lack of conservation) of energy-momentum at a vertex. Below is a list of properties for basic scenario we have observed in Monte-Carlo generators used for testing the code.

This list will serve as a declaration for convention of `HepMC` filling, which the interface should be able to interpret correctly.

- **4-momentum conservation** is assumed for all vertices in the event record where `PHOTOS` is expected to act.
- **Status codes:** only information on whether a given particle is incoming, outgoing or intermediate will be used. We assume the codes used will be 0, 1 or 2 like in HEPEVT. All other codes will be treated as equivalent to the status 2.
- **Pointers at a vertex** are assumed to be bi-directional. That is, it is possible to traverse the record structure from mother to daughter and from daughter to mother along the same path.

Extensions/Exceptions to these specifications are handled in some cases. We will call them options for conventions of event record filling.

- Vertices like $\tau^{\pm} \to \tau^{\pm}$ and $\tau^{\mp} \to \tau^{\mp}\gamma$ where the decaying particle flavor is among its decay products will prevent `PHOTOS` being invoked.
- If there is 4-momentum non-conservation[4] in the vertex, `PHOTOS` will not be invoked too. A special kinematic correcting method to remove smaller inconsistencies resulting e.g. from rounding errors is available too.
- As in the `FORTRAN` cases, we expect that new types of conventions for filling the event record will appear, because of physics motivated requirements. Unfortunately, the resulting options do not always guarantee an algebraically closed structure. Host program-specific patches may need to be defined for `PHOTOS`. Debugging can then be time consuming, and will need to be repeated for every new case.

In the future, an important special case of event record's filling, with information extracted from experimentally observed events (e.g. $Z \to \mu^+\mu^-$ modified later to $Z \to \tau^+\tau^-$) should be allowed. Obviously, a new type (or types) of `HepMC` filling will then appear.

A good example is the evolution of `PYTHIA`. While in version 8.108 the status codes for our example processes took the values 0, 1 or 2 only (in the part of the record important for `PHOTOS`), other values were already present in version 8.135. As a consequence the status code for otherwise nicely decaying particles was not always 2 anymore. We have introduced a change in the file PhotosEvent.cxx to adjust. After the change the program should work for all previous cases as before, changes like this one are usually difficult to validate and

[4]For details see Appendix C.5 for details.

complicated tests are necessary. One could investigate if instead of changes on the side of the PHOTOS algorithm a different choice of input for PYTHIA would not be a more appropriate solution, but in this case we choose to adopt our algorithm.

We have yet to decide on bar code conventions. Our programs, TAUOLA and PHOTOS, supplement the event record with new particle entries carrying bar codes with values starting from 10001. That is the choice resulting from our use of HepMC methods and defaults.

2.3 Interface to the Event Record of FORTRAN

In principle it would be rather simple to completely rewrite PHOTOS to C++. However to profit from numerical tests, for the time being the numerical core of PHOTOS is still left in FORTRAN. The C++ part of the code searches the whole event for suitable HepMC vertices for the generation of bremsstrahlung. Once such a vertex is found it is copied to an internal event record which is called PH_HEPEVT (this is effectively the same structure as HEPEVT; the event record of FORTRAN). The FORTRAN code of PHOTOS is then executed. The data structure passed in this way is rather simple. Only a single vertex consisting of the decaying particle along with its mothers and daughters is passed. Information on mothers will be useful in future for the calculation of process dependent kernels.

A description of the interface to the remaining FORTRAN parts of the code is given in Appendix A.

3 Design

3.1 Classes and Responsibilities

The choice of splitting the source code into three main modules allows the separation of the FORTRAN related code from the abstract C++ interface and the concrete implementation of the interface created for the appropriate event record.

- PHOTOS FORTRAN **Interface**
  This part of the code provides an interface to the FORTRAN library of PHOTOS. In particular, a wrapper for the routine which invokes the processing of a single branch. Parts of the interface code are still left in FORTRAN, but can be rather easily rewritten to C++. The most important part of this module, the PH_HEPEVT_Interface_ class, is already implemented in C++. This class is responsible for writing the decay branch to be processed into the internal event record PH_HEPEVT. For further details regarding the interface to FORTRAN common blocks and routines see Appendix A.

- PHOTOS **C++ Interface**
  This is an abstract interface to the event record. The class PhotosEvent contains information regarding the whole event structure, while PhotosParticle stores all information regarding a single particle. All particles used by the interface are located in the event in the form of a list of PhotosParticle objects. The last class located here, PhotosBranch, contains information regarding elementary branching to be processed by PHOTOS.

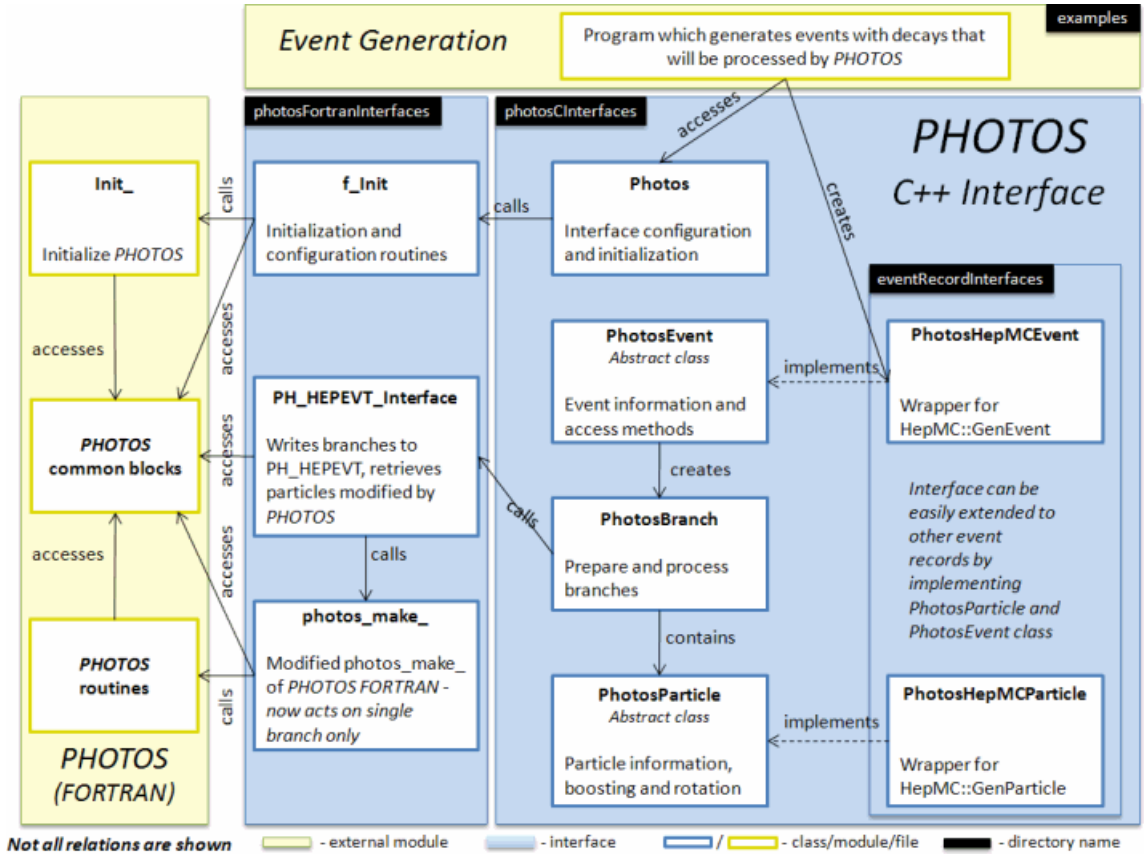


Figure 1: `PHOTOS` C++ interface class relation diagram

- **Event Record Interface**
  This contains the event record implementation classes. All classes stored here represent the implementation of specific event record interfaces and are responsible for reading, traversing and writing to the event record structure. Only the `PhotosEvent` and `PhotosParticle` classes must be implemented. The `HepMC` event record interface is implemented through `PhotosHepMCEvent` and `PhotosHepMCParticle`. These classes are similar to the analogous `TAUOLA` [?] event record classes. An example of a minimalistic interface to an event record has been provided through the classes `PhotosHEPEVTEvent` and `PhotosHEPEVTParticle`[5]. They can be used as a template for a new interface to any other event record[6].

3.2 Directory Structure

- **src/eventRecordInterfaces/** - source code for classes which interface with event records. Currently only the `HepMC` interface is located here.
  Classes:
  - **PhotosHepMCEvent** - interface to HepMC::GenEvent objects.
  - **PhotosHepMCParticle** - interface to HepMC::GenParticle objects.
  - **PhotosHEPEVTEvent** - interface to the event structure of the `HEPEVT` format.

[5]This interface is the only way of implementing NLO corrections in programs based on `HEPEVT` event record

[6]Thanks to the polymorphism, abstract part of the algorithm is well separated from the specific event record implementations. It even allows simultaneous use of several distinct event record implementations.

  - **PhotosHEPEVTParticle** - interface to a single particle from the HEPEVT event record.

- **src/photosCInterfaces/** - source code for the general PHOTOS interface.
  Classes:

  - **Photos** - controls the configuration and initialization of PHOTOS.
  - **PhotosEvent** - abstract base class for event information.
  - **PhotosParticle** - abstract base class for particles in the event.
  - **PhotosBranch** - contains one PhotosParticle, and pointers to its mothers and daughters. The filtering of branchings to be tried by PHOTOS for photons emission is done here.

- **src/photosFortranInterfaces/** - interface to PHOTOS FORTRAN routines and common blocks.
  Files:

  - **f_Init** - contains a wrapper for the PHOTOS FORTRAN routines for initialization.
  - **PH_HEPEVT_interface** - contains a wrapper for accessing the PH_HEPEVT common block.
  - **photos_make.f** - contains FORTRAN routines to be migrated to C++ rather soon.
  - **forW-ME.f** - c FORTRAN routines to calculate matrix element in W decay, own programming style.
  - **forZ-ME.f** - FORTRAN routines to calculate matrix element in Z decay, own programming style.

- **src/utilities/** - source code for utilities.
  Classes:

  - **Log** - general purpose logging class that allows filtering out output messages of PHOTOS C++ Interface and tracks statistics for each run.
  - **PhotosRandom** - random number generator taken from PHOTOS FORTRAN and rewritten to C++.

- **src/photos-fortran/** - PHOTOS FORTRAN core rewritten to C++. Since version 3.54 PHOTOS has been fully rewritten to C++ and located in its own namespace Photospp[7]. From the algorithmic point of view full compatibility[8] with the FORTRAN version is kept. The appropriate descriptions remain vaild; in publications as well as in the code, now in C++.

- **examples/** - examples of different PHOTOS C++ Interface uses.

  - **photos_hepevt_example** - stand alone example with a simple $e^+e^- \to \tau^+\tau^-$ event written in HEPEVT format and then processed by PHOTOS.
  - **photos_standalone_example** - the most basic example which loads pre-generated events stored in a file in HepMC format which are then processed by PHOTOS.
  - **single_photos_gun_example** - an example of using the processOne method to process only selected particles within the event record.

[7]This means that no part of the code is shared with old PHOTOS FORTRAN and both versions can be loaded simultanously without the risk of one version overwriting the options of the other.

[8]The resulting modules are however not interchangeable and the program will not function if the PHOTOS FORTRAN library is loaded instead of code encapsulated in **src/photos-fortran/** directory.

- **photos_pythia_example** - an example of $e^+e^- \to Z \to \mu^+\mu^-$ events generated by `PYTHIA 8.1` and processed by `PHOTOS`. The analysis is done using `MC-TESTER`.
- **tauola_photos_pythia_example** - an example of `TAUOLA` linked with `PYTHIA 8.1`. The decay chain is processed by `PHOTOS` and then analyzed with `MC-TESTER`.

- **include/** - directory for the header files.
- **lib/** - directory for the compiled libraries.
- **documentation/** - contains doxygen documentation and this latex file.

3.3 Algorithm Outline

An overview of the algorithm for the `PHOTOS C++ Interface` is given below. For clarity, the example assumes that the processed event is stored in the `HepMC` event record structure.

The first step is creation of a `PhotosHepMCEvent` object from a `HepMC::GenEvent` event record. After that, the `process()` method should be executed by the user's code[9], invoking the following process:

1. The `HepMC` event record is traversed and a list of all decaying particles is created.
2. Each particle is checked and if the resulting branching is a self-decay[10] it is skipped.
3. For each remaining particle a branching including the particle's mothers and daughters is created. Special cases consisting of mothers and daughters but without intermediate particle to be decayed are also added to the list of branchings.
4. Branchings are filtered out again, this time with the user's choice of processes to be skipped by the photon adding algorithm.
5. Each branching is processed by `PHOTOS` separately:
   (a) The branching is written to `PH_HEPEVT`
   (b) The `PHOTOS` photon adding algorithm is invoked
   (c) The resulting branching is taken back from `PH_HEPEVT` and any changes made by `PHOTOS` to the already existing particles and their whole decay trees are integrated into the event record.
   (d) Finally, the created photons are added to the event record.

The underlying HepMC::GenEvent is hence modified with the insertion of photons.

[9] Instead of creating a `PhotosHepMCEvent` and processing the whole event, a user may want to execute `Photos::processParticle(...)` or `Photos::processBranch(...)` for branching or branch, where `PHOTOS` is expected to perform its tasks. For details see Appendix C.3.

[10] A history entry in the event record, like $Z \to Z$ or $\tau \to \tau$.

4 Extensibility

The purpose of the present version of the C++ interface to PHOTOS is to enable all the functionality of its FORTRAN predecessor to be available for the HepMC data structure. Some methods for improved initialization are already introduced in this version. The new program functionality has been prepared to enable further extensions in the future. Let us briefly discuss some of these points.

4.1 PHOTOS Extensions

In this paper we have presented an algorithm as it is today. That is, a version which is compatible with the one implemented in FORTRAN.

- As can be seen from the text presented, we have prepared the structure for the implementation of channel dependent matrix elements. This work, requiring special modifications to the FORTRAN code and applicable one at a time only can be simply integrated all together into the C++ version. This is not urgent. We are not convinced if such complication is worth the inconvenience for the user as it requests more strict control of the event record content. The gain of precision is not that important because the precision is already quite good. This is why we have not incorporated these auxiliary upgrades to C++ already now. Instead, as an intermediate step, we developed validation techniques for the program. These are required prior to a re-write of the numerically sensitive part of the program (which is algorithm-wise rather simple) anyway.

- Methods devised to check the content of HepMC as used by PHOTOS are described in Appendix C.4. They need to be used whenever PHOTOS processes events from a new generator e.g. upgraded versions of PYTHIA, which may fill the event record in an unexpected way. Experience gained from many years of developing and maintaining the algorithm have shown, that this is the most demanding task; the necessity to adapt to varying physics and technical input of the event record pose a multitude of problems. The nature of these difficulties cannot be predicted in advance.

- For the sake of debugging we have introduced new control methods and ones which activate internal printouts of the FORTRAN part of the code. The routine PHLUPA [**?**] can be activated to verify how an event is constructed/modified, and to investigate energy momentum (non-) conservation or other inconsistencies. This is quite convenient, for example, for tracing problems in the information passed to PHOTOS.

- Numerical stability is another consideration; it cannot be separated from physics constraints. The condition $E^2 - p^2 = m^2$ may be broken because of rounding errors. However, due to intermediate particles with substantial widths, the on mass shell condition may not be applicable. PHOTOS may be adapted to such varying conditions, but it requires good interaction with users. The protection which removes inconsistencies in the event record may be a source of unexpected difficulties in the other cases. At present, such methods are left in the FORTRAN part of the algorithm, which will be gradually reviewed and migrated to C++.

4.2 Event Record Interface

In the times of FORTRAN, the PHOTOS interface used an internal event structure which was based on HEPEVT, adding to it (understood as a data type) an extra variable defining the status of particles with respect to QED Bremsstrahlung. In some cases, like $\tau \to l\nu_l\nu_\tau$, bremsstrahlung was already generated earlier by other generator, and PHOTOS should not be activated on such decays. At present, instead, a set of initialization methods is prepared as described in Appendix C.1.

There is definite room for improvement. For example if the vertex $q\bar{q} \to l^{\pm}l^{\mp}g$ is encountered (note the presence of g in the final state), the interface could 'on the fly' add an intermediate Z into the record and enable PHOTOS on the temporarily constructed decay branching, $Z \to l^{\pm}l^{\mp}$.

Internally, in the FORTRAN part of PHOTOS, the data structures based on HEPEVT: PH_HEPEVT and PHOEVT, are used, but they store only a single elementary decay. At this step in the program evolution it might not be the most elegant solution, but it prevents the need to redo many of the FORTRAN era benchmarks.

5 Testing

One of the most important parts of the PHOTOS project are its physics oriented tests. Several domains of physics tests should be mentioned. Users interested in precision simulations of Z or W decay will find the papers [**?**, **?**, **?**] most interesting. There, careful comparison with the first order matrix element and confirmation of the agreement is concluded first. For Z decays, comparisons with Monte Carlo program based on exclusive exponentiation with up to second order matrix element is possible and was performed on some benchmark distributions. Inclusion of a correcting weight was found to be numerically less important than absence of the second order matrix element in the YFS exponentiation scheme. In these comparisons the Monte Carlo programs from LEP times [**?**, **?**] were used. In numerical tests MC-TESTER [**?**] was used. The advantage of the method is that C++ and FORTRAN program results can be easily compared[11].

For inclusive calculations, FSR radiative corrections are at the one permille level. For semi-inclusive cross sections, such as the total rates of Z decay for events where the hardest photon energy (or two hardest photon energies) exceed 1 GeV in the Z rest frame, differences between results from PHOTOS (PHOTOS without the matrix element correcting weight) and YFS based generators of the first or second order were also at the 0.2 % level. On the other hand, if two hard photons were requested and invariant masses constructed from leptons and hard photons were monitored, the level of differences exceeded 30 %. However, even in this region of phase space PHOTOS, without the correcting weight, performs better[12] than programs based on exponentiation and the first order matrix element only.

This conclusion needs to be investigated if realistic experimental cuts are applied. Fortunately the necessary programs are available for Z decay. In the case of W decay, second order Monte Carlo generators supplemented with exponentiation are not available at this moment.

[11] We thank Andy Buckley for checking numerically that our conclusions on the first order exact YFS exponentiation results extend to the programs presently used at the LHC such as SHERPA and HERWIG++.

[12] In the phase space region were only one hard photon is tagged this conclusion seems to depend on the variant of exponentiation in use, [**?**] or [**?**].

For users interested in the simulation of background for Higgs searches at the LHC and for any other applications where two hard photon configurations are important, studies based on the comparison with a double photon matrix element are of interest. For PHOTOS Monte Carlo such tests were initiated in refs. [?, ?, ?]. Finally, users interested in low energy processes where the underlying physics model for photon emission cannot be controlled by theory sufficiently well (scalar QED may be considered only as the starting point), will profit from [?, ?]. In all cases it is important that the program generation cover the full phase-space and that there are no approximations in phase-space. As in the FORTRAN version, the code features approximation in the kernel. In some cases the process dependent complete first order kernel is available but it is not always installed in the public version, because the resulting approximation is numerically unimportant and requires control of the spin state for the decaying particle. This requires better control of the event record than what was available in FORTRAN. At present such an option is prepared (see Section 2.3) for W and Z decays. It is also available for the two body decays of scalar into scalars. Then, exact mean exact with respect to scalar QED only.

The main purpose of the present paper is program documentation. That is why we also need to cover the program tests required to guarantee its proper installation. The physics tests discussed above do not guarantee that the program will perform well on a particular platform and installation. Tests and debugging of the installation are necessary too. If the content of the event record is non-standard or rounding errors are large, the performance of PHOTOS will deteriorate.

The first check after installation of PHOTOS is whether some energy momentum non conservation appears. Such offending events should be studied before PHOTOS and after PHOTOS is run to modify them. If it is impossible to understand why inconsistencies for energy momentum non conservation were created by PHOTOS, the authors should be contacted. Sometimes monitoring how an event is constructed inside the FORTRAN part of the code may be useful. For that purpose the steering of monitoring[13] is available from the C++ part of the code. In practice only rather advanced users will profit from the printouts. However, it may be sent to the authors and help to quickly identify the cause of the problems.

The next step in benchmarking relies on comparisons with benchmark distributions. At present, we store these tests with the help of ROOT [?] and our MC-TESTER program [?].

5.1 MC-TESTER Benchmark Files

Over years of development of the TAUOLA and PHOTOS programs a certain level of the automation of tests was achieved. It was found that monitoring all the invariant mass distributions which can be constructed out of a given decay represent a quite restrictive but easy to implement test. Finding the relative fraction of events for each distinct final state complemented that test and is implemented now in the public version of MC-TESTER. We have applied this method for PHOTOS too. In this case, some soft final state particles have to be ignored because we are bound to work with samples which otherwise would exhibit properties of unphysical infrared regulators (see Section 6.1 of ref [?] for more details). For the most popular decays the benchmarks are collected on our project web page [?]. In our distribution, we have collected numerical results in the directory `examples/testing` and its subdirectories: `Htautau, ttbar, Wenu, Wmunu, Zee and Zmumu`. Each of them includes an appropriate initialization files for the particular run of PYTHIA. Numerical results from long runs of MC-TESTER based

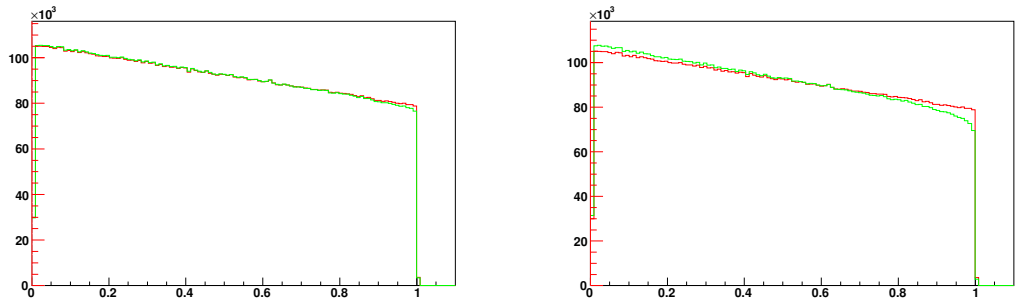
[13] See Appendix C.4 for the command `Log::LogPhlupa(int from, int to)`

tests are stored for reference[14]. At present, our choice of tests is oriented toward the LHC user and radiative corrections in decay of W's, Z's and Higgs particles. Most of the users of low energy experiments use the FORTRAN version of the code, which is why our tests and examples for the C++ version are not geared toward low energy applications.

5.2 Results

In principle, for the algorithm of photon(s) construction, the C++ interface of PHOTOS introduces nothing new with respect to the version of PHOTOS as available in FORTRAN. That is why the tests which are collected in [?] are not recalled here, they were only reproduced and found to be consistent with the ones for old PHOTOS FORTRAN. However the algorithm for combining a modified branching, including the added photon(s), to the rest of the event tree was rewritten. Examples of spin dependent observables are of more interest because they test this new part of the code. The π energy spectrum in the decaying Z boson rest-frame is the first example. The $\pi^{\pm}$ originates from $\tau^{\pm} \to \pi^{\pm}\nu$ decay and, as was already shown a long time ago [?], its energy spectrum is modified by bremsstrahlung both in τ and Z decays. The net bremsstrahlung effect is similar to the one of e.g. Z polarization. In Fig. 2 this result is reproduced.

Let us now turn to tests using observables which are sensitive to transverse spin effects. For that purpose we study the decay chain: $H \to \tau^+\tau^-$, $\tau^{\pm} \to \rho^{\pm}\nu_{\tau}$, $\rho^{\pm} \to \pi^{\pm}\pi^0$, where PHOTOS may act on any of the branchings listed above. An inappropriate action of the C++ part of PHOTOS could result in faulty kinematics, manifesting itself in a failure of energy-momentum conservation checks of the event record or faulty spin correlation sensitive distributions. However, as we can see from Fig. 3, the distributions (as they should be) remain nearly identical to the ones given in [?, ?]. The emission of soft and/or collinear photons to the τ^+ or τ^- does not change the effects of spin correlations. The kinematical effects of hard, non collinear photons are responsible for dips in the acoplanarity distributions at 0, π and 2π.

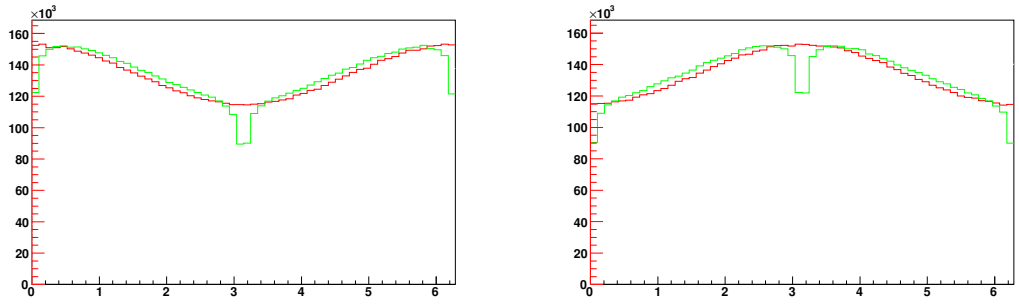


(a) bremsstrahlung from τ^+ decay only

(b) bremsstrahlung from Z, τ^+ and τ^- decays

Figure 2: Bremsstrahlung effects for longitudinal spin observables for the cascade decay: $Z \to \tau^+\tau^-$, $\tau^{\pm} \to \pi^{\pm}\nu$. π^+ energy spectrum in the Z rest-frame is shown. The red line is for bremsstrahlung switched off and green (light grey) when its effect is included. In the left plot, bremsstrahlung is in τ^+ decay only. In the right plot, bremsstrahlung from Z and $\tau^{\pm}$ decays is taken into account. These plots have been prepared using a custom UserTreeAnalysis of MC-TESTER. They can be recreated with the test located in the examples/testing/Ztautau directory, see examples/testing/README-plots for technical details. Results are consistent with Fig. 5 of Ref. [?].

[14]Details on the initialization for the runs are given in README-benchfiles.



(a) selection C (b) selection D

Figure 3: Bremsstrahlung effects for transverse spin observable: The distribution of the acoplanarity angle of oriented planes spanned respectively on the $\pi^+\rho^+$ and $\pi^-\rho^-$ momenta is shown. The distribution is defined in the rest frame of the $\rho^+\rho^-$ pair for the scalar Higgs decay chain $H \to \tau^+\tau^-$, $\tau^\pm \to \rho^\pm\nu_\tau$, $\rho^\pm \to \pi^\pm\pi^0$. `PHOTOS` is used to generate bremsstrahlung. The red curve indicates the distribution when bremsstrahlung effects are ignored and for the green curve (light grey) only events with bremsstrahlung photons of energy larger than 1 GeV in the H rest frame are taken. For the definition of selections C and D see. [**?**, **?**]. These plots have been created using a custom `UserTreeAnalysis` of `MC-TESTER`. They can be recreated by the test located in the `examples/testing/Htautau` directory, see `examples/testing/README-plots` for technical details.

In Ref. [**?**] a discussion of the systematic errors for the measurement of the Z cross section at the LHC is presented. One of the technical tests of our software is to obtain Fig. 1b of that paper. In our Fig. 4 we have reproduced that plot using `PYTHIA 8.1` and `PHOTOS`. Qualitatively, the effect of FSR QED bremsstrahlung is in the two cases quite similar.

In Fig. 4 we present a plot of the bare electron pair (which means electrons are not combined with the collinear photons that accompany them) from Z decay with and without `PHOTOS`. It is similar to the plots shown for `Horace` or `Winhac`; see Refs. [**?**, **?**] and related studies. One should bear in mind that this is again a technical test with little direct application to physics. As explained in the figure caption, the LHC production process was used.

We have also checked that `PHOTOS` works for $t\bar{t}$ events and the simulations explained in [**?**] can be repeated. For this purpose we have produced $t\bar{t}$ pairs in pp collisions at 14 TeV center-of-mass energy. We have produced rates for events with zero, one or at least two photons of energy above 0.001 of the $t\bar{t}$ pair mass (energies are calculated in the hard scattering frame). Results are given in the following table which is constructed from $gg \to t\bar{t}$ events only:

Final state	Branching Ratio (%) ± Statistical Errors (%)
$\widetilde{t}t$	99.0601 ± 0.0315
$\widetilde{t}t\gamma$	0.9340 ± 0.0031
$\widetilde{t}t\gamma\gamma$	0.0060 ± 0.0002

10 million events were generated and a slightly modified version of `MC-TESTER`'s `LC-analysis` from Ref. [**?**] was used for calculation of the event rates[15]. `ROOT` files for differential distributions are collected in the directory `examples/testing/ttbar`.

[15] We do not supplement the list of final state particles with the second mother, in contrast to the choice used in `LC-analysis` from Ref. [**?**].

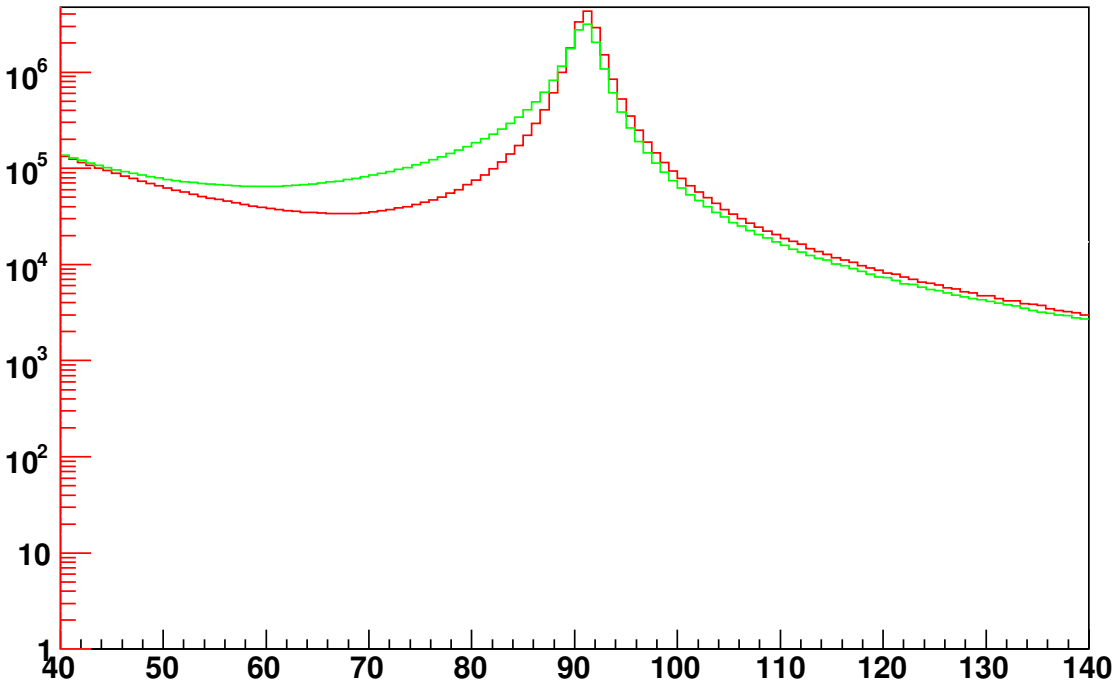


Figure 4: Distribution of the bare e^+e^- invariant mass. The green curve (light grey) represents results when final state bremsstrahlung is generated (with the help of `PHOTOS` Monte Carlo). For the red curve FSR bremsstrahlung is absent. The event sample is generated with the help of `PYTHIA 8.1`. The simulation of pp collisions at 14 TeV center-of-mass energy is performed. The Z/γ mediated hard process is used. This plot has been created using a custom `UserTreeAnalysis` of `MC-TESTER`. It can be recreated with the test located in the `examples/testing/Zee` directory, see `examples/testing/README-plots` for technical details.

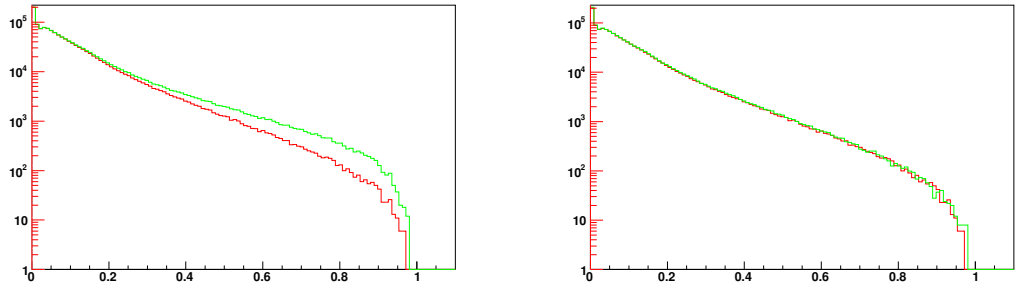
5.3 Tests Relevant for Physics Precision

Let us now turn to an example of the test for the two photon final state configuration. We compare (i) KKMC [**?**] with exponentiation and second order matrix element (CEEX2), (ii) KKMC with exponentiation and first order matrix element (CEEX1) and finally (iii) the results of `PHOTOS` with exponentiation activated. As one can see from the table below, the rates coincide for the three cases up to two permille for the event configurations where zero, one or at least two photons of energy above 1 GeV are accompanying the $\mu^+\mu^-$ pair.

Agreement at this level is not seen in the differential distributions, see Fig. 5. For example the spectrum of the two photon mass is quite different between the first and second order exponentiation result. This is of potential interest for background simulations for $H \to \gamma\gamma$. In contrast, the difference between the results from `PHOTOS` and CEEX2 are much smaller.

Decay channel	Branching Ratio ± Rough Errors (100M event samples)		
	KKMC CEEX2 (%)	KKMC CEEX1 (%)	`PHOTOS` exp. (%)
$Z^0 \to \mu^+\mu^-$	83.9190 ± 0.0092	83.7841 ± 0.0092	83.8470 ± 0.0092
$Z^0 \to \gamma\mu^+\mu^-$	14.8152 ± 0.0038	14.8792 ± 0.0039	14.8589 ± 0.0039
$Z^0 \to \gamma\gamma\mu^+\mu^-$	1.2658 ± 0.0011	1.3367 ± 0.0012	1.2940 ± 0.0011

PHOTOS exploits the first order matrix element in a better way than exponentiation. As a consequence it reproduces terms resulting in second order leading logarithms. This observation is important not only for the particular case of Z decay but for the general case of double bremsstrahlung in any decay as well.



(a) CEEX2: red; CEEX1: green

(b) CEEX2: red; PHOTOS: green

Figure 5: The spectrum of the $\gamma\gamma$ invariant mass in $Z \to \mu^+\mu^-$ decay. Events with two hard photons, both of energy above 1 GeV in the Z rest frame are taken. Comparisons are shown for CEEX2 and CEEX1 (left plot), and CEEX2 and PHOTOS (right plot). The prediction from PHOTOS is clearly superior for applications aimed at the simulation of Higgs boson backgrounds. In the case of solutions based on YFS exponentiation, the second order matrix element must be taken into account. Fig. 5(b) was obtained from our example, `examples/testing/Zmumu`, after adaptation of the center-of-mass energy (91.17 GeV) and test energy threshold (1 GeV). Samples of 100 million events were used. See `examples/testing/README-plots` for technical details. Reference ROOT files for KKMC CEEX samples are, however, created outside of the distribution.

The numerical results collected here provide part of the program benchmarks. They are of limited but nonetheless of some physics interest as well. PHOTOS provides only one step in the simulation chain: bremsstrahlung in decays or final state bremsstrahlung. One can ask the question of whether such a specialized unit is of interest and whether it is not better to provide the complete chain for "truth physics simulation" as a single simulation package. Obviously, this will depend on particular needs. Final state QED corrections can be separated from the remaining genuine electroweak corrections and in particular the initial state QED bremsstrahlung from quarks. One should bear in mind, that final state bremsstrahlung needs to be disentangled from detector acceptance dependencies and is not of interest in itself. This must be done e.g. to measure the properties of weak bosons.

One should bear in mind that even for QED FSR alone, the discussion of the physics precision of the simulation result requires further checks. In the case of Z decays, Refs. [**?**, **?**] may not be enough. With increasing precision, the estimation of uncertainty becomes dependent on the particular choice of details for the experimental cuts. Comparisons of different calculations become important too. A good example of such work in the conext of other measurements can be found in Refs. [**?**, **?**]; for the simulation of FSR QED corrections, the Monte Carlo programs collected for PHOTOS generator tests are probably enough. A discussion of QED initial final state interference may follow the strategy presented in [**?**]. There, the question of experimental cuts must be included as well. Once these steps are completed, discussion of complete electroweak corrections in the context of realistic observables should be simplified.

Genuine weak corrections have to be taken into account separately. Such solutions may be possible, if together with the PHOTOS `Interface`, weak corrections are provided for example with the help of the TAUOLA `Interface`. A discussion of the complete electroweak corrections, as shown in Fig. 1a of [**?**], is not the purpose of our document. Let us point out however

that electroweak non-QED corrections can be in principle installed into the PYTHIA 8.1 + PHOTOS simulation with the help of the e.g. TAUOLA interface [?]. But for such a solution to be precise, further work is needed [?].

Sizeable initial state QED corrections are usually embodied in units simulating parton showers. This may need some experimental analysis as well. Recently experimental data from LEP1 times were revisited by the DELPHI collaboration [?]. Tension between data and the theoretical description is mentioned. This may mean that the description of initial state QED bremsstrahlung at the LHC will need to be re-investigated with the help of LHC data as well. That is also why it might be useful to keep initial state QED, final state QED and their interference corrections in separate modules.

We leave such estimations of the theoretical uncertainty for Z and W production at the LHC to the forthcoming research, where a realistic choice of experimental conditions and observables will become an integrated part.

6 Summary and outlook

We have presented a new version of PHOTOS Monte Carlo. The part of PHOTOS which operates on event records is now rewritten into C++ and uses the HepMC event record format for input and output. The physics performance of the program is the same as that of the FORTRAN/HEPEVT version, but better steering options are introduced. Also, when an elementary decay is to be modified by PHOTOS, it is first transformed to its rest frame. The z-axis is orientated along the decaying particle's mother's direction, as seen in this rest frame. Such modification is necessary to calculate process dependent kernels featuring the complete first order matrix element. If the interest will arise, the appropriate kernels explained in Refs. [?, ?, ?] can be installed into the C++ version of PHOTOS. The necessary information is extracted from the event record and can be used. At present some parts of the algorithm are left in FORTRAN. This remaining FORTRAN part of the code can be recoded to C++ rather easily, but the necessary numerical tests are time consuming. That is why we have left these changes for later releases.

Finally let us point to ref. [?]. Thanks to this work, for LHC applications in Z, W decays, PHOTOS Monte Carlo systematic error was established at 0.3%, even 0.2% for the case of matrix element correction activated. The estimation is valid for complete final state radiative corrections, not for photonic bremsstrahlung alone.

Acknowledgements

Useful discussions with P. Golonka during the early stage of project development and discussions with members of the ATLAS and CMS collaborations and the LCG/Genser team are acknowledged.

Partial support by the Polish-French collaboration no. 10-138 within IN2P3 through LAPP Annecy and during the final completion of this work is also acknowledged.

A Appendix: Interface to PHOTOS FORTRAN

This appendix is addressed to developers of the interface, and special users interested in advanced options of PHOTOS. The discussed below COMMON blocks of FORTRAN were used in the program up to version 3.53 only. Starting from version 3.54 some of these COMMON blocks were preserved, as the struct objects of C. The following ones may be of some interest for the user: PHOCOP, PHOKEY, PHOSTA, PHLUPY.

Names of variables and structs are not modified with respect to FORTRAN, the C++ definition style is adapted, small letters are used and underscore is added at the end of structs names. Common HEPEVT is preserved for use of example and future applications, where the main program is in FORTRAN. The interface is available through the classes PhotosHEPEVTEvent.h and PhotosHEPEVTParticle.h. Internally in C++ Photos code, the HEPEVT-like data type is used.

A.1 Common Blocks

In the following let us list the common blocks of PHOTOS which are accessed from C++.

PHOCOP coupling constant and related parameters.

- **ALPHA** *double* coupling constant α_{QED}.
- **XPHCUT** *double* minimal energy (in units of half of the decaying particle's mass) for photons to be explicitly generated.

PHOKEY keys and parameters controlling the algorithm options.

- **FSEC** *double* internal variable for algorithm options, the default is FSEC=1.0 .
- **FINT** *double* maximum interference weight.
- **EXPEPS** *double* technical parameter, blocks crude level high photon multiplicity from configurations less probable than EXPEPS, the default is 10^{-4}.
- **INTERF** *bool* key for interference, the matrix element weight.
- **ISEC** *bool* switch for double bremsstrahlung generation.
- **ITRE** *bool* switch for bremsstrahlung generation up to a multiplicity of 4.
- **IEXP** *bool* switch for exponentiation mode.
- **IFTOP** *bool* switch for photon emission in top pair production in quark (gluon) pair annihilation.
- **IFW** *bool* switch for leading effects of matrix element in leptonic W decays.

PHLUPY debug messages handling.

- **IPOIN** *int* messages above this value will not be displayed.
- **IPOINM** *int* messages below this value will not be displayed.

PH_PHOQED supplement for the PH_HEPEVT event record to block emission from some particles.

QEDRAD[10000] *bool* flag enabling a `FORTRAN` users to block emission from particles stored in PH_HEPEVT. Not used in our interface.

PHOSTA Status information.

STATUS[10] *int* Status codes for the last 10 errors/warnings that occurred.

PHPICO π value definition.

PI *double* π.

TWOPI *double* $2 * \pi$.

A.2 Routines

In the following let us list routines which are called from the interface.

PHODMP prints out the content of HEPEVT.
Return type: *void*
Parameters: none

PHOTOS_MAKE `FORTRAN` part of interface.
Return type: *void*
Parameters:

1. *int id* ID of particle from which `PHOTOS` starts processing. In the C++ case the importance of this parameter is limited as only one branch is in HEPEVT at a time.

PHCORK initializes kinematic corrections.
Return type: *void*
Parameters:

1. *int modcor* type of correction. See Ref. [**?**] for details.

IPHQRK enables/blocks (2/1) emission from quarks.
Return type: *int*
Parameters: *int*

IPHEKL enables/blocks (2/1) emission in: $\pi^0 \to \gamma e^+ e^-$.
Return type: *int*
Parameters: *int*

B Appendix: User Guide

B.1 Installation

`Photos C++ Interface` is distributed in a form of an archive containing source files and examples. Currently only the Linux and Mac OS[16] operating systems are supported: other systems may be supported in the future if sufficient interest is found.

The main interface library uses `HepMC` [?] (version 2.03 or later) and requires that either its location has been provided or option of compilation without `HepMC` has been chosen during the configuration step. This is sufficient to compile the interface and to run the simple, stand-alone example.

However, in order to run the more advanced examples located in the `/examples` directory, `HepMC` is required. It is also required to install:

- `ROOT` [?] version 5.18 or later
- `PYTHIA 8.1` [?] or later. `PYTHIA 8.1` must be compiled with `HepMC` 2.xx so that the `PYTHIA` library `hepmcinterface` exists.
- `MC-TESTER` [?, ?] version 1.24 or later. Do not forget to type `make libHepMCEvent` after compilation of `MC-TESTER` is done.
- `TAUOLA` [?] version 1.0.5 or later. `TAUOLA` must be compiled with `HepMC`.

In order to compile the `PHOTOS` C++ Interface:

- Execute `./configure` with the additional command line options:

  `--with-hepmc=<path>` provides the path to the `HepMC` installation directory. One can also set the `HEPMCLOCATION` variable instead of using this directive. To compile the interface without `HepMC` use `--without-hepmc`

  `--prefix=<path>` provides the installation path. The `include` and `lib` directories will be copied there if `make install` is executed later. If none has been provided, the default directory for installation is `/usr/local`.

- Execute `make`
- Optionally, execute `make install` to copy files to the directory provided during configuration.

The `PHOTOS` C++ interface will be compiled and the `/lib` and `/include` directories will contain the appropriate libraries and include files.

In order to compile the examples, compile the `PHOTOS` C++ interface, enter the `/examples` directory and:

- Execute `./configure` to determine which examples can be compiled. Additional paths can be provided as command line options:

[16] For this case LCG configuration scripts explained in Appendix B.2 have to be used.

`--with-pythia8=<path>` provides the path to the `Pythia8` installation directory. One can set the `PYTHIALOCATION` variable instead of using this directive. This path is required for all examples and tests.

`--with-mc-tester=<path>` provides the path to the `MC-TESTER` installation directory (the `libHepMCEvent` must be compiled as well, see Ref. [**?**] for more details). One can set the `MCTESTERLOCATION` variable instead of using this directive. This path is required for all additional examples and tests. This option implies that `ROOT` has already been installed (since it is required by `MC-TESTER`). The `ROOT` directory `bin` should be listed in the variable `PATH` and the `ROOT` libraries in `LD_LIBRARY_PATH`.

`--with-tauola=<path>` provides the path to the `TAUOLA` installation directory. One can set the `TAUOLALOCATION` variable instead of using this directive. This path is required for additional examples only.

- Execute `make`

If neither `HepMC`, `Pythia8` nor `MC-TESTER` are accessible, the most basic example will be nonetheless provided. The `/examples` directory will contain the executable files for all examples which can be compiled and linked.

On `CERN` platforms, ready to use software is located in the `/afs/cern.ch/sw/lcg/` directory. To set the appropriate paths, the scripts `platform/afs.paths.sh` and `platform/afs.paths.csh` are prepared and should be executed before `./configure`. For details see `README` in the `PHOTOS` main directory.

B.2 LCG configuration scripts; available from version 3.1

For our project still another configuration/automake system was prepared for use in LCG/Genser project[17] [**?**, **?**].

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.

B.3 Elementary Tests

The most basic test which should be performed, for our custom examples but also for a user's own generation chain, is verification that the interface is installed correctly, photons are indeed added by the program and that energy momentum conservation is preserved[18].

[17] We have used the expertise and advice of Dmitri Konstantinov and Oleg Zenin in organization of configuration scripts for our whole distribution tar-ball as well. Thanks to this choice, we hope, our solution will be compatible with the ones in general use.

[18] We have performed such tests for all choices of the `HepMC` event record obtained from `PYTHIA 8.1` and `PYTHIA 8.135` processes and listed in the paper. Further options for initializations (parton shower hadronization or QED bremsstrahlung on/off etc.) were also studied. This was a necessary step in our program

In principle, these tests have to be performed for any new hard process and after any new installation. This is to ensure that information is passed from the event record to the interface correctly and that physics information is filled into `HepMC` in the expected manner. Misinterpretation of the event record content may result in faulty `PHOTOS` operation.

B.4 Executing Examples

Once elementary tests are completed one can turn to the more advanced ones. The purpose is not only to validate the installation but to demonstrate the interface use.

The examples can be run by executing the appropriate `.exe` file in the `/examples` directory. In order to run some more specific tests for the following processes: $H \to \tau^+\tau^-$, $e^+e^- \to t\bar{t}$, $W \to e\nu_e$, $W \to \mu\nu_\mu$, $Z \to e^+e^-$, $Z \to \mu\mu$ or $Z \to \tau^+\tau^-$, $K_0^S \to \pi\pi$, the main programs residing in the subdirectories of `/examples/testing` should be executed. In all cases the following actions have to be performed:

- Compile the `PHOTOS` C++ Interface.
- Check that the appropriate system variables are set. Execution of the script `/configure.paths.sh` usually can perform this task; the configuration step announces this script.
- Enter the `/examples/testing` directory. Execute `make`. Modify test.inc if needed.
- Enter the sub-directory for the particular process of interest and execute `make`.

The appropriate `.root` files as well as `.pdf` files generated by `MC-TESTER` will be created inside the chosen directory. One can execute 'make clobber' to clean the directory. One can also execute 'make run' inside the `/examples/testing` directory to run all available tests one after another. Changes in source code can be partly validated in this way. Most of the tests are run using the executable `examples/testing/photos_test.exe`. The $K_0^S \to \pi\pi$, $H \to \tau^+\tau^-$ and $Z \to \tau^+\tau^-$ examples require `examples/testing/photos_tauola_test.exe` to be run. After generation, `MC-TESTER` booklets will be produced, comparisons to the benchmark files will be shown. A set of benchmark `MC-TESTER` root files have been included with the interface distribution. They are located in the subdirectories of `examples/testing/`. Note that for the $W \to e\nu_e$, $W \to \mu\nu_\mu$ and $Z \to \mu\mu$ examples, differences higher than statistical error will show. This is because photon symmetrization was used in the benchmark files generated with `KKMC`, and not in the ones generated with `PHOTOS`. In the case of `KKMC` the generated photons are strictly ordered in energy. In the case of `PHOTOS` they are not. Nonetheless, on average, the second photon has a smaller energy than the one written as the first in the event record.

The comparison booklets can be useful to start new work or simply to validate new versions or new installations of the `PHOTOS` interface.

In Appendix C, possible modifications to the example's settings are discussed. This may be interesting as an initial step for user's physics studies. The numerical results of some of these tests are collected in Section 5.2 and can be thus reproduced by the user.

development.

B.5 How to Run PHOTOS with Other Generators

If a user is building a large simulation system she or he may want to avoid integrating into it our configuration infrastructure and load the libraries only. For that purpose our stand-alone example **examples/photos_standalone_example.exe** is a good starting point.

In order to link the libraries to the user's project, both the static libraries and shared objects are constructed. To use the PHOTOS interface in an external project, additional compilation directives are required. For the static libraries:

- add -I<PhotosLocation>/include at the compilation step,
- add <PhotosLocation>/lib/libPhotosCxxInterface.a <PhotosLocation>/lib/libPhotosFortra at the linking step of your project.

For the shared objects:

- add -I<PhotosLocation>/include at the compilation step,
- add -L<PhotosLocation>/lib along with -lPhotosCxxInterface -lPhotosFortran at the linking step.
- PHOTOS libraries must be provided for the executable; e.g. with the help of LD_LIBRARY_PATH.

<PhotosLocation> denotes the path to the PHOTOS installation directory. In most cases it should be enough to include within a users's program Photos.h and PhotosHepMCEvent.h (or any other header file for the class implementing abstract class PhotosEvent) With that, Photos class can be used for configuration and PhotosHepMCEvent for event processing.

B.5.1 Running PHOTOS C++ Interface in FORTRAN environment

For backward-compatibility with HEPEVT event record, an interface has been prepared allowing PHOTOS C++ Interface to be invoked from the FORTRAN project. An example photos_hepevt_example.f has been prepared to demonstrate how PHOTOS can be initialized and executed from FORTRAN code. Since PHOTOS works in C++ environment, photos_hepevt_example_interface.cxx must be introduced to invoke PHOTOS.

Since version 3.54 PHOTOS is fully in C++ and initialization can no longer be performed from FORTRAN code through the use of common blocks.

C Appendix: User Configuration

C.1 Suppress Bremsstrahlung

In general, PHOTOS will attempt to generate bremsstrahlung for every branching point in the event record. This is of course not always appropriate. Already inside the FORTRAN part of the new PHOTOS, bremsstrahlung is prevented for vertices involving gluons or quarks (with the exception of top quarks).

This alone is insufficient. By default we suppress bremsstrahlung generation for vertices like $l^{\pm} \to l^{\pm}\gamma$ because a "self-decay" is unphysical. We cannot request that all incoming and/or outgoing lines are on mass shell, because it is not the case in cascade decays featuring intermediate states of sizeable width. If a parton shower features a vertex with $l^{\pm} \to l^{\pm}\gamma$ with the virtuality of the incoming $l^{\pm}$ matching the invariant mass of the outgoing pair then the action of PHOTOS at this vertex will introduce an error. This is prevented by forbidding bremsstrahlung generation at vertices where one of the decay products has a flavor which matches the flavor of an incoming particle.

Some exceptions to the default behavior may be necessary. For example in cascade decays, the vertex $\rho \to \rho\pi$ may require the PHOTOS algorithm to be activated.

Methods to re-enable these previously prevented cases or to prevent generation in special branches have been introduced and are presented below.

For the user's convenience the following configuration methods are provided:

- `Photos::suppressBremForDecay(daughterCount, motherID, d1ID, d2ID, ...)`
  The basic method of channel suppression. The number of daughters, PDGID of the mother and the list of PDGIDs of daughters must be provided. There is no upper limit to the number of daughters. If a decay with the matching pattern is found, PHOTOS will skip the decay.

- `Photos::suppressBremForDecay(0, motherID)`
  When only the PDGID of the mother is provided, PHOTOS will skip all decay channels of this particle. Note that the number of daughters is to be provided, but set for this case to zero.

- `Photos::suppressBremForBranch(daughterCount, motherID, d1ID, d2ID, ...)`
  `Photos::suppressBremForBranch(0, motherID)`
  The usage of these methods is similar to the previous functions. The difference is that PHOTOS will skip not only the corresponding channel, but also all consecutive decays of its daughters, making PHOTOS skip the entire branch of decays instead of just one.

- `Photos::suppressAll()` All branchings will be suppressed except those that are forced using the methods described in the next section.

- **Example:**
  ```
  Photos::suppressBremForDecay(3, 15, 16, 11, -12);
  Photos::suppressBremForDecay(2, -15, -16, 211);
  Photos::suppressBremForDecay(0, 111);
  ```
  If the decays $\tau^- \to \nu_\tau e^- \bar{\nu}_e$ or $\tau^+ \to \bar{\nu}_\tau \pi^+$ are found, they will be skipped by PHOTOS. *In addition, all decays of π^0 will also be skipped. Note, that the minimum number of parameters that must be provided is two - the number of daughters (which should be zero if suppression for all decay channels of the particle is chosen) and the mother PDGID.*

  ```
  Photos::suppressBremForBranch(2, 15, 16, -213);
  ```
  When the decay $\tau^- \to \nu_\tau \rho^-$ is found, it will be skipped by PHOTOS *along with the decays of ρ^- (in principle also ν_τ) and all their daughters. In the end, the whole decay tree starting with $\tau^- \to \nu_\tau \rho^-$ will be skipped.*

In future, an option to suppress a combination of consecutive branches may be introduced. For example if bremsstrahlung in leptonic τ decays is generated by programs prior to PHOTOS,

and the decay is stored in HepMC as the cascade $\tau^{\pm} \to W^{\pm}\nu$, $W^{\pm} \to l^{\pm}\nu$, PHOTOS must be prevented from acting on both vertices, but only in cases when they are present one after another. One can also think of another PHOTOS extension for the following type of special case: if a vertex $q\bar{q} \to l^{\pm}l^{\mp}$ is found then it should not be ignored but passed for generation as $q\bar{q} \to Z \to l^{\pm}l^{\mp}$ where the intermediate Z is created internally for PHOTOS.

C.2 Force PHOTOS Processing

Forcing PHOTOS to process a branch can be used in combination with the suppression of all branches i.e. to allow selection of only a particular processes for bremsstrahlung generation.

Forced processing using the methods below has higher priority than the suppression described in the previous section, therefore even if both forcing and suppressing of the same branch or decay is done (regardless of order), the processing will not be suppressed.

- `Photos::forceBremForDecay(daughterCount, motherID, d1ID, d2ID, ...)`
  `Photos::forceBremForDecay(0, motherID)`
  The usage of this routine is similar to `Photos::suppressBremForDecay(...)` described in the previous section. If a decay with the matching pattern is found, PHOTOS will be forced to process the corresponding decay, even if it was suppressed by any of the methods mentioned in the previous section.

- `Photos::forceBremForBranch(daughterCount, motherID, d1ID, d2ID, ...)`
  `Photos::forceBremForBranch(0, motherID)`
  The usage is similar to the above functions. The difference is that PHOTOS will force not only the corresponding channel, but also all consecutive decays of its daughters, making PHOTOS process the entire branch of decays instead of just one.

- **Example:**
  ```
  Photos::suppressAll();
  Photos::forceBremForDecay(4, 15, 16, -211, -211, 211);
  Photos::forceBremForDecay(2, -15, -16, 211);
  Photos::forceBremForBranch(0, 111);
  ```
  Since suppression of all processes is used, only the listed decays will be processed, these are $\tau^- \to \nu_\tau\pi^-\pi^-\pi^+$, $\tau^+ \to \bar{\nu}_\tau\pi^+$ and all instances of the decay of π^0 and its descendants.

C.3 Use of the processParticle and processBranch Methods

In Section 3.3 the algorithm for processing a whole event record is explained and is provided through the `process()` method. To process a single branch in the event record, in a way independent of the entire event, a separate method is provided.

- `Photos::processParticle(PhotosParticle *p)`
  The main method for processing a single particle. A pointer to a particle must be provided. Pointers to mothers and daughters of this particle should be accessible through this particle or its event record. From this particle a branch containing its mothers and daughters will be created and processed by PHOTOS.

- `Photos::processBranch(PhotosParticle *p)`
  Usage is similar to the above function. When a pointer to a particle is provided, `PHOTOS` will process the whole decay branch starting from the particle provided.

An example, `single_photos_gun_example.c`, is provided in the directory `/examples` showing how this functionality can be used to process the decay of selected particles. $Z^0 \to \tau^+\tau^-$ decays are generated and the event record is traversed searching for the first τ^- particle in the event record. Instead of processing the whole event, only the decay of τ^- is processed by `PHOTOS`.

C.4 Logging and Debugging

This section describes the basic functionality of the logging and debugging tool. For details on its content we address the reader to comments in the `/src/utilities/Log.h` header file.

Let us present however some general scheme of the tool's functionality. The `PHOTOS` interface allows control over the amount of message data displayed during program execution and provides a basic tool for memory leak tracking. The following functions can be used from within the user's program after including the `Log.h` file:

- `Log::Summary()` - Displays a summary of all messages.
- `Log::SummaryAtExit()` - Displays the summary at the end of a program run.
- `Log::LogInfo(bool flag)`
  `Log::LogWarning(bool flag)`
  `Log::LogError(bool flag)`
  `Log::LogDebug(int s, int e)`
  `Log::LogAll(bool flag)`
  Turns the logging of *info*, *warning*, *error* and *debug* messages on or off depending on the flag value being true or false respectively. In the case of *debug* messages - the range of message codes to be displayed must be provided. By default, only *debug* messages (from 0 to 65535) are turned off. If the range is negative ($s > e$) *debug* messages won't be displayed. The last method turns displaying all of the above messages on and off.
- `Log::LogPhlupa(int from, int to)`
  Turns logging of debug messages from the `FORTRAN` part of the program on and off. Parameters of this routine specify the range of debug codes for the `phlupa_` routine.

At present only the following debug messages can be printed:

- Debug(0) - seed used by the random number generator
- Debug(1) - which type of branching was found in `HepMC` (regular or a case without an intermediate particle, for details see `PhotosBranch.cxx`)
- Debug(700) - execution of the branching filter has started
- Debug(701) - branching is forced
- Debug(702) - branching is suppressed
- Debug(703) - branching is processed (i.e. passed to the filter)

- Debug(2) - execution of the branching filter was completed
- Debug(1000) - the number of particles sent to and retrieved from PHOTOS FORTRAN

The option `Log::SetWarningLimit(int limit)` results in only the first '`limit`' warnings being displayed. The default for `limit` is 100. If `limit`=0 is set, then there are no limits on the number of warnings to be displayed.

The memory leak tracking function allows checking of whether all memory allocated within PHOTOS Interface is properly released. However, using the debug option significantly increases the amount of time needed for each run. Its use is therefore recommended for debugging purposes only. In order to use this option modify `make.inc` in the main directory by adding the line:
`DEBUG = -D"_LOG_DEBUG_MODE_"`
Recompile the interface. Now, whenever the program is executed a table will be printed at the end of the run, listing all the pointers that were not freed, along with the memory they consumed. If the interface works correctly without any memory leaks, one should get an empty table.

It is possible to utilize this tool within a user's program; however there are a few limitations. The debugging macro from "Log.h" can create compilation errors if one compiles it along with software which has its own memory management system (e.g. ROOT). To make the macro work within a user's program, ensure that `Log.h` is the last header file included in the main program. It is enough to compile the program with the `-D"_LOG_DEBUG_MODE_"` directive added, or `#define _LOG_DEBUG_MODE_` placed within the program before inclusion of the `Log.h` file[19].

C.5 Other User Configuration Methods

The following auxiliary methods are prepared. They are useful for initialization or are introduced for backward compatibility.

- `Photos::setRandomGenerator(double (*gen)())` *installed in* PHOTOS 3.52
  Replace random number generator used by `Photos`. The user provided generator must return a `double` between 0 and 1. `Photos::setRandomGenerator(NULL)` will reset the program back to the default generator, which is a copy of RANMAR[?, ?].
- `Photos::setSeed(int iseed1, int iseed2)`
  Set the seed values for our copy of the random number generator RANMAR [?, ?].
- `Photos::maxWtInterference(double interference)`
  Set the maximum interference weight. The default 2 is adopted to decays where at most two charged decay products are present[20].
- `Photos::setInfraredCutOff(double cut_off)`
  Set the minimal energy (in units of decaying particle mass) for photons to be explicitly generated.

[19] Note that `Log.h` does not need to be included within the user's program for the memory leak tracking tool to be used only for the PHOTOS interface.

[20] For the decays like $J/\psi \to 5\pi^+5\pi^-$ higher value, at least equal to the number of charged decay products, should be set. The algorithm performance will slow down linearly with the maximum interference weight but all simulation results will remain unchanged.

- `Photos::setAlphaQED(double alpha)`
  Set the coupling constant, alpha QED.

- `Photos::setInterference(bool interference)`
  A switch for interference, matrix element weight.

- `Photos::setDoubleBrem(bool doub)`
  Set double bremsstrahlung generation.

- `Photos::setQuatroBrem(bool quatroBrem)`
  Set bremsstrahlung generation up to a multiplicity of 4.

- `Photos::setExponentiation(bool expo)`
  Set the exponentiation mode.

- `Photos::setCorrectionWtForW(bool corr)`
  A switch for leading effects of the matrix element (in leptonic W decays)

- `Photos::setMeCorrectionWtForScalar(bool corr)`
  A switch for complete effects of the matrix element (in scalar to 2 scalar decays) *installed in* `PHOTOS 3.3`. *At present tests are missing.*

- `Photos::setMeCorrectionWtForW(bool corr)`
  A switch for complete effects of the matrix element (in leptonic W decays) *installed in* `PHOTOS 3.2` Because of lack of reinitialization in a particular decay channel this option can be used for the fixed decay channel, and either for W+ or for W-, also for fixed (not strongly varying) wirtuality. This option will be elaborated further.

- `Photos::setMeCorrectionWtForZ(bool corr)`
  A switch for complete effects of the matrix element (in leptonic Z decays) *installed in* `PHOTOS 3.1`

- `Photos::initializeKinematicCorrections(int flag)`
  Initialize kinematic corrections.

- `Photos::forceMassFrom4Vector(bool flag)`
  By default, for all particles used by `PHOTOS`, mass is re-calculated and $\sqrt{E^2 - p^2}$ is used. If `flag=false`, the particle mass stored in the event record is used. The choice may be important for the control of numerical stability in case ov very light stable particles, but may be incorrect for decay products themselves of non-negligible width.

- `Photos::forceMass(int pdgid, double mass)` *installed in* `PHOTOS 3.4`
  For particles of PDGID (or -PDGID) to be processed by `PHOTOS`, mass value attributed by user will be used instead of the one calculated from 4-vector. Note that if both `forceMass` and `forceMassFromEventRecord` is used for the same PDGID, the last executed function will take effect. Up to version 3.51, option active if `forceMassFrom4Vector = true` (default). From version 3.52, option works regardless of setting of `forceMassFrom4Vector`.

- `Photos::forceMassFromEventRecord(int pdgid)` *installed in* `PHOTOS 3.4`
  For particles of PDGID (or -PDGID to be processed by `PHOTOS`, mass value taken from the event record will be used instead of the one calculated from 4-vector. Note that if both `forceMass` and `forceMassFromEventRecord` is used for the same PDGID, the last executed function will take effect.

  Up to version 3.51 option active if `forceMassFrom4Vector = true` (default). From version 3.52, option works regardless of setting of `forceMassFrom4Vector`.

- `Photos::createHistoryEntries(bool flag, int status)` *installed in* `PHOTOS 3.4`
  If set to `true`, and if event record format allows, `Photos` will store history entries consisting of particles before processing. History entries will have status code equal `status`. The value of `status` will also be added to the list of status codes ignored by `Photos` (see `Photos::ignoreParticlesOfStatus`)[21]. An example is provided in `photos_pythia_example.cxx`.

- `Photos::ignoreParticlesOfStatus(int status)` Decay products with the status code `status` will be ignored in check of momentum conservation and will not be passed to algorithm generating bremsstrahlung.

- `Photos::deIgnoreParticlesOfStatus(int status)` Removes `status` from the list of status codes created with `Photos::ignoreParticlesOfStatus`).

- `bool Photos::isStatusCodeIgnored(int status)` Returns `true` if `status` is on the list of ignored status codes.

- `Photos::setMomentumConservationThreshold(double momentum_conservation_threshold)`
  The default value is 0.1 (in `MEV/GEV`, depending on the units used by `HepMC`). If larger energy-momentum non conservation is found then in the vertex, photon generation is skipped. At present, for the evaluation of non conservation the standard method of `HepMC` is used.

- `Photos::iniInfo()`
  The printout performed with `Photos::initialize()` will exhibit outdated information once the methods listed above are applied. The reinitialized data can be printed with the help of `Photos::iniInfo()` method. The format as in `Photos::initialize()` will be used.

C.6 Creating Advanced Plots and Custom Analysis

In Section 5.2, we have presented results of a non-standard analysis performed by `MC-TESTER`. Figure 4 has been obtained using a custom `UserTreeAnalysis` located in the `ZeeAnalysis.C` file residing in the `examples/testing/Zee` directory. This file serves as an example of how custom analysis can be performed and how new plots can be added to the project with the help of `MC-TESTER`.

The basic `MC-TESTER` analysis contains methods used by pre-set examples in the subdirectories of `examples/testing` directory to focus on at most one or two sufficiently hard photons from all the photons generated by `PHOTOS`. Its description and usage have already been documented in [**?**]. The content of `ZeeAnalysis.C` is identical to the default `UserTreeAnalysis` of `MC-TESTER` with the only addition being a method to create the previously mentioned plot.

In order to create the $t\bar{t}$ example, an additional routine had to be added to `photos_test.c`. Since `MC-TESTER` is not designed to analyze processes involving multiple incoming particles, we have used a method similar to that previously used in the `FORTRAN` examples `LC_Analysis` mentioned in [**?**], Section 6.1. This routine, `fixForMctester`, transforms the $XY \to t\bar{t}$ process to the $100 \to t\bar{t}$ process, where the momentum of the special particle 100 is $X + Y$. With this modification, `MC-TESTER` can be set up to analyze the particle 100 in order to get a desirable result.

[21] In case of `HepMC`, creates copies of all particles on the list of outgoing particles in vertices where photon was added and will be added at the end of the list.

For more details regarding the plots created for this documentation, see `README-plots` located in `examples/testing/` directory.