LPAIR++

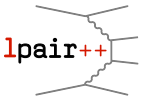
0.2

Generated by Doxygen 1.8.6

Thu Jan 8 2015 18:41:26

Contents

1 Principles



This Monte Carlo generator, based on the LPAIR code developed in the early 1990s by J. Vermaseren *et al*[4], allows to compute the cross-section and to generate events for the $\gamma\gamma \to \ell^+\ell^-$ process in high energy physics.

The main operation is the integration of the matrix element (given as a subset of a Process object) performed by *Vegas*, an importance sampling algorithm written in 1972 by G. P. Lepage[2].

2 Deprecated List

Global MCGen::LaunchGeneration ()

This method is to be suppressed since the events generation can now be launched one event at a time using the *GenerateOneEvent* method

3 Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

4 Data Structure Index

4.1 Data Structures

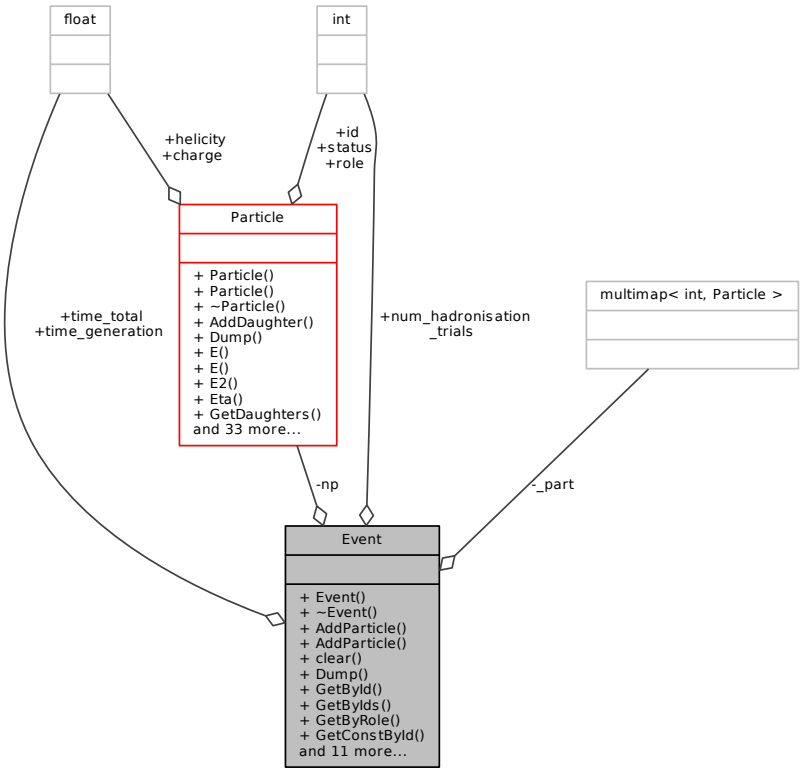
Here are the data structures with brief descriptions:

5 Data Structure Documentation

5.1 Event Class Reference

Kinematic information on the particles in the event.

Collaboration diagram for Event:



Public Member Functions

- int AddParticle (Particle part_, bool replace_=false)

  Add a particle to the event.
- int AddParticle (int role_, bool replace_=false)

  Creates a new particle in the event, with no kinematic information but the role it has to play in the process.
- void clear ()

  Empties the whole event content.
- void Dump (bool stable_=false)
- Particle * GetById (int id_)

  Gets one particle by its unique identifier in the event.
- ParticlesRef GetByIds (std::vector< int > ids_)

  Gets a vector of particles by their unique identifier in the event.
- ParticlesRef GetByRole (int role_)

  Gets a list of particles by their role in the event.
- const Particle **GetConstById** (int id_) const
- Particles **GetConstMothers** (Particle *part_) const
- Particles **GetConstParticles** () const
- ParticlesRef GetDaughters (Particle *part_)

  Gets a vector containing all the daughters from a particle.
- ParticlesRef GetMothers (Particle *part_)
- Particle * GetOneByRole (int role_)
- ParticlesRef GetParticles ()

  Gets a vector of particles in the event.
- std::vector< int > GetRoles () const
- ParticlesRef GetStableParticles ()

  Gets a vector of stable particles in the event.
- int NumParticles () const

  Number of particles in the event.
- Event & operator= (const Event &)

  Copies all the relevant quantities from one Event object to another.
- void Store (std::ofstream *, double weight_=1.)

  Gets the LHE block for this event.

Data Fields

- int num_hadronisation_trials

  Number of trials before the event was "correctly" hadronised.
- float time_generation

  Time needed to generate the event at parton level.
- float time_total

  Time needed to generate the hadronised (if needed) event.

Private Attributes

- ParticlesMap _part
- Particle * np

5.1.1 Detailed Description

Class containing all the information on the in- and outgoing particles' kinematics

5.1.2 Member Function Documentation

5.1.2.1 int Event::AddParticle (**Particle** part_, bool replace_ = `false`)

Sets the information on one particle in the process

Parameters

in	*part_*	The Particle object to insert or modify in the event
in	*replace_*	Do we replace the particle if already present in the event or do we append another particle with the same role ?

Returns

- 1 if a new Particle object has been inserted in the event
- 0 if an existing Particle object has been modified
- -1 if the requested role to edit is undefined or incorrect

5.1.2.2 int Event::AddParticle (int role_, bool replace_ = `false`)

Parameters

in	*role_*	The role the particle will play in the process
in	*replace_*	Do we replace the particle if already present in the event or do we append another particle with the same role ?

Returns

- 1 if a new Particle object has been inserted in the event
- 0 if an existing Particle object has been modified
- -1 if the requested role to edit is undefined or incorrect

5.1.2.3 void Event::Dump (bool stable_ = `false`)

Dumps all the known information on every Particle object contained in this Event container in the output stream

Parameters

in	*stable_*	Do we only show the stable particles in this event ?

5.1.2.4 **Particle*** Event::GetById (int id_)

Returns the pointer to the Particle object corresponding to a unique identifier in the event

Parameters

in	*id_*	The unique identifier to this particle in the event

Returns

A pointer to the requested Particle object

5.1.2.5 ParticlesRef Event::GetByIds (std::vector< int > ids_) `[inline]`

Returns the pointers to the Particle objects corresponding to the unique identifiers in the event

Parameters

in	*ids_*	The unique identifiers to the particles to be selected in the event

Returns

A vector of pointers to the requested Particle objects

5.1.2.6 ParticlesRef Event::GetByRole (int role_)

Returns the list of pointers to the Particle objects corresponding to a certain role in the process kinematics

Parameters

in	*role_*	The role the particles have to play in the process

Returns

A vector of pointers to the requested Particle objects

5.1.2.7 ParticlesRef Event::GetDaughters (**Particle** * part_) [inline]

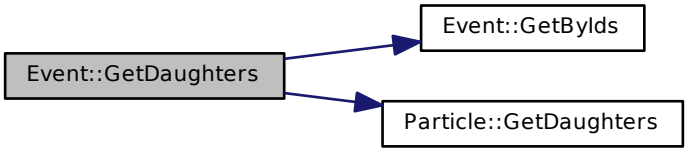
Parameters

in	*part_*	The particle for which the daughter particles have to be retrieved

Returns

A Particle objects vector containing all the daughters' kinematic information

Here is the call graph for this function:



5.1.2.8 ParticlesRef Event::GetMothers (**Particle** * part_) [inline]

Returns the pointer to the mother particle of any given Particle object in this event

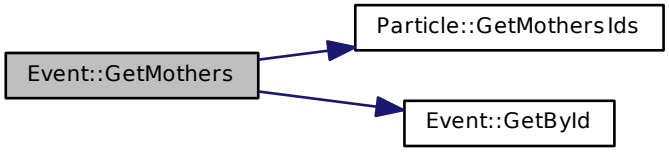
Parameters

in	*part_*	The pointer to the Particle object from which we want to extract the mother particle

Returns

A pointer to the mother Particle object

Here is the call graph for this function:



5.1.2.9 **Particle*** Event::GetOneByRole (int role_) [inline]

Returns the first Particle object in the particles list whose role corresponds to the given argument

Parameters

in	*role_*	The role the particle has to play in the event

Returns

A Particle object corresponding to the first particle found in this event

Here is the call graph for this function:

5.1.2.10 ParticlesRef Event::GetParticles ( )

Returns

A vector containing all the pointers to the Particle objects contained in the event

5.1.2.11 std::vector<int> Event::GetRoles ( ) const

Gets a list of roles for the given event (really process-dependant for the central system)

Returns

A vector of integers corresponding to all the roles the particles can play in the event

5.1.2.12 ParticlesRef Event::GetStableParticles ( )

Returns

A vector containing all the pointers to the stable Particle objects contained in the event

5.1.2.13 int Event::NumParticles ( ) const [inline]

Returns

The number of particles in the event, as an integer

5.1.2.14 void Event::Store (std::ofstream * , double weight_ = 1.)

Returns an event block in a LHE format (a XML-style) with all the information on the particles composing this event

Parameters

in	*weight_*	The weight of the event

Returns

A string containing the kinematic quantities for each of the particles in the event, formatted as the LHE standard requires. Stores in a file (raw format) all the kinematics on the outgoing leptons

Parameters

in	*weight_*	The weight of the event

5.1.3 Field Documentation

5.1.3.1 ParticlesMap Event::_part [private]

List of particles in the event, mapped to their role in this event

5.1.3.2 **Particle*** Event::np [private]

Empty particle returned to the get-ers if no particle matches the requirements

5.1.3.3 float Event::time_generation

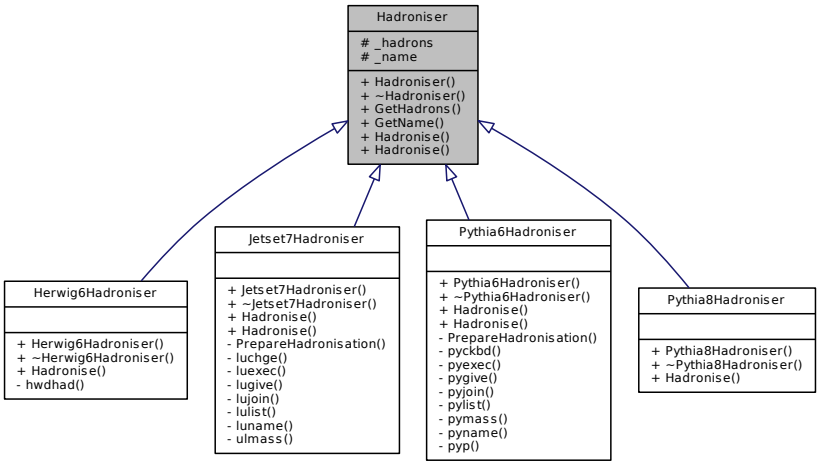
The time took by the generator to build the event without hadronising it, in seconds

5.1.3.4 float Event::time_total

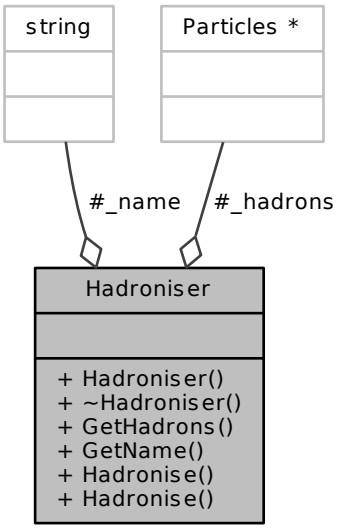
The time took by the generator to build and hadronise the event, in seconds

5.2 Hadroniser Class Reference

Inheritance diagram for Hadroniser:



Collaboration diagram for Hadroniser:



Public Member Functions

- Particles GetHadrons ()
- std::string GetName ()

  Returns the human-readable name of the hadroniser used.
- virtual bool Hadronise (Particle *part_)

  Main caller to hadronise a single particle.
- virtual bool Hadronise (Event *ev_)

  Hadronises a full event.

Protected Attributes

- Particles * _hadrons

  List of hadrons produced by this hadronisation process.
- std::string _name

  Name of the hadroniser.

5.2.1 Detailed Description

Class template to define any hadroniser as a general object with defined methods

Author

Laurent Forthomme laurent.forthomme@uclouvain.be

Date

January 2014

5.2.2 Member Function Documentation

5.2.2.1 Particles Hadroniser::GetHadrons ( ) [inline]

Gets the full list of hadrons (as Particle objects) produced by the hadronisation

Returns

A vector of Particle containing all the hadrons produced

5.2.2.2 virtual bool Hadroniser::Hadronise (**Particle** * part_) [inline], [virtual]

Parameters

in	*part_*	The Particle object which will be hadronised

Returns

A boolean stating whether or not the hadronisation occured successfully

Reimplemented in Pythia6Hadroniser, and Jetset7Hadroniser.

5.2.2.3 virtual bool Hadroniser::Hadronise (**Event** * ev_) [inline], [virtual]

Launches the hadroniser on the full event information

Parameters

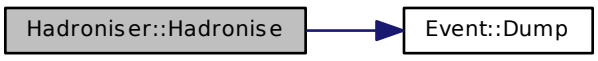
in,out	*ev_*	The event to hadronise

Returns

A boolean stating whether or not the hadronisation occured successfully

Reimplemented in Pythia6Hadroniser, Herwig6Hadroniser, Jetset7Hadroniser, and Pythia8Hadroniser.

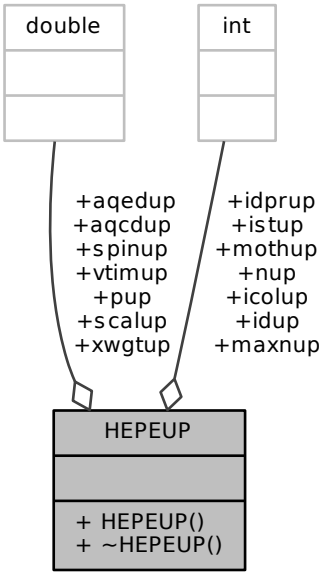
Here is the call graph for this function:



5.3 HEPEUP Class Reference

User-process event information.

Collaboration diagram for HEPEUP:



Public Member Functions

- **HEPEUP** (const int nup_=500)

Data Fields

- double aqcdup

  QCD coupling $\alpha_{\rm QCD}$ used for this event.
- double aqedup

  QED coupling $\alpha_{\rm QED}$ used for this event.
- int * icolup [2]

  Index for the colour flow line passing through the colour (resp. anti-colour) of the particle.
- int idprup

  ID of the process in this event.
- int * idup

  Particle ID according to the Particle Data Group convention.
- int * istup

  Status code.
- int * mothup [2]

  Index of first and last mother.
- int nup

  Number of particle entries in this event.
- double * pup [5]

  Lab-frame momentum of the particle, in GeV.
- double scalup

  Scale of the event in GeV, as used for the calculation of PDFs.

- double * spinup

  Cosine of the angle between the spin-vector of the particle and the 3-momentum of the decaying particle, in the lab frame.
- double * vtimup

  Invariant lifetime $c\tau$ in mm.
- double xwgtup

  Event weight.

Static Public Attributes

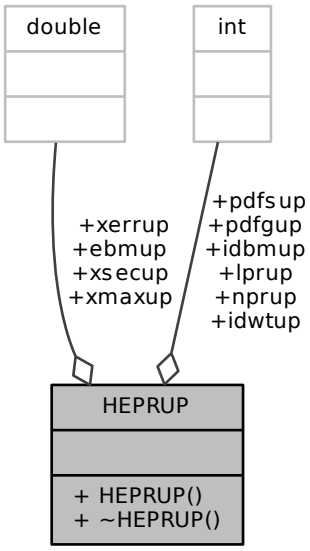
- static const int maxnup = 500

  Maximum number of particle entries.

5.4 HEPRUP Class Reference

Generic user-process interface for events generator.

Collaboration diagram for HEPRUP:



Public Member Functions

- **HEPRUP** (const int nprup_=1)

Data Fields

- double ebmup [2]

  Energy in GeV of the beam 1 and 2 particles.
- int idbmup [2]

  ID of the beam 1 and 2 particles according to the Particle Data Group convention.

- int **idwtup**
- int * **lprup**
- int **nprup**
- int pdfgup [2]

  Author group for beam 1 and 2, according to PDFLIB.
- int pdfsup [2]

  PDF set ID for beam 1 and 2, according to PDFLIB.
- double * **xerrup**
- double * **xmaxup**
- double * **xsecup**

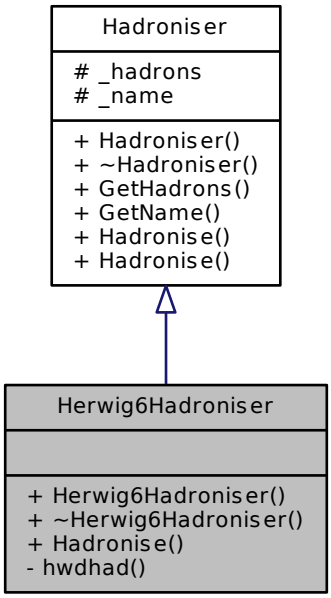
5.4.1 Detailed Description

User-process run information

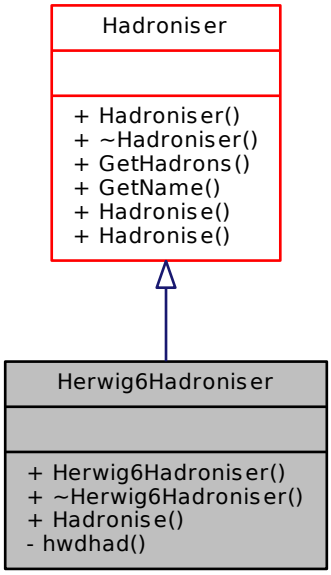
5.5 Herwig6Hadroniser Class Reference

Herwig6 hadronisation algorithm.

Inheritance diagram for Herwig6Hadroniser:



Collaboration diagram for Herwig6Hadroniser:



Public Member Functions

- Particles GetHadrons ()
- std::string GetName ()

  Returns the human-readable name of the hadroniser used.
- virtual bool Hadronise (Particle *part_)

  Main caller to hadronise a single particle.
- bool Hadronise (Event *ev_)

  Hadronises a full event.

Protected Attributes

- Particles * _hadrons

  List of hadrons produced by this hadronisation process.
- std::string _name

  Name of the hadroniser.

Static Private Member Functions

- static void **hwdhad** ()

5.5.1 Member Function Documentation

5.5.1.1 Particles Hadroniser::GetHadrons ( ) [inline], [inherited]

Gets the full list of hadrons (as Particle objects) produced by the hadronisation

Returns

A vector of Particle containing all the hadrons produced

5.5.1.2 virtual bool Hadroniser::Hadronise (**Particle** * part_) [inline], [virtual], [inherited]

Parameters

in	*part_*	The Particle object which will be hadronised

Returns

A boolean stating whether or not the hadronisation occured successfully

Reimplemented in Pythia6Hadroniser, and Jetset7Hadroniser.

5.5.1.3 bool Herwig6Hadroniser::Hadronise (**Event** * ev_) [virtual]

Launches the hadroniser on the full event information

Parameters

in,out	*ev_*	The event to hadronise

Returns

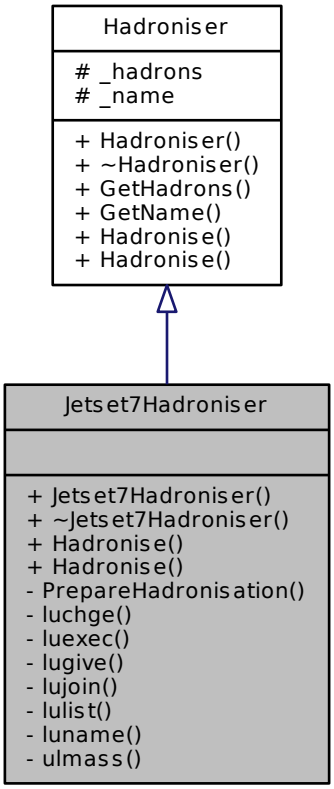
A boolean stating whether or not the hadronisation occured successfully

Reimplemented from Hadroniser.

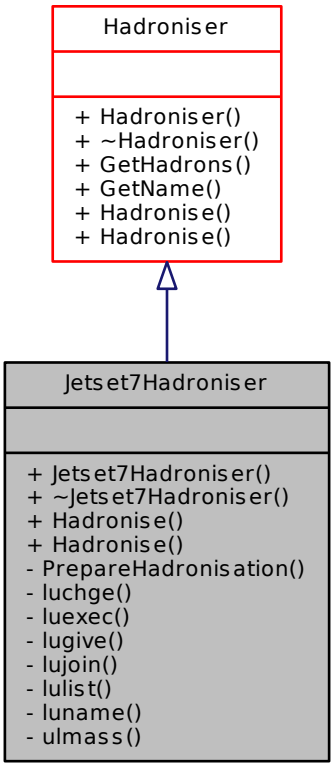
5.6 Jetset7Hadroniser Class Reference

Jetset7 hadronisation algorithm.

Inheritance diagram for Jetset7Hadroniser:



Collaboration diagram for Jetset7Hadroniser:



Public Member Functions

- Particles GetHadrons ()
- std::string GetName ()

  Returns the human-readable name of the hadroniser used.
- bool Hadronise (Particle *part_)

  Main caller to hadronise a single particle.
- bool Hadronise (Event *ev_)

  Hadronises a full event.

Protected Attributes

- Particles * _hadrons

  List of hadrons produced by this hadronisation process.
- std::string _name

  Name of the hadroniser.

Private Member Functions

- bool **PrepareHadronisation** (Event *ev_)

Static Private Member Functions

- static float luchge (int pdgid_)
- static void luexec ()
- static void lugive (const std::string &line_)
- static void lujoin (int njoin_, int ijoin_[2])
- static void lulist (int mlist_)

  List an event, jet or particle data, or current parameter values.
- static std::string luname (int pdgid_)
- static double ulmass (int pdgid_)

5.6.1 Member Function Documentation

5.6.1.1 Particles Hadroniser::GetHadrons ( ) [inline], [inherited]

Gets the full list of hadrons (as Particle objects) produced by the hadronisation

Returns

A vector of Particle containing all the hadrons produced

5.6.1.2 bool Jetset7Hadroniser::Hadronise (**Particle** * part_) [virtual]

Parameters

in	*part_*	The Particle object which will be hadronised

Returns

A boolean stating whether or not the hadronisation occured successfully

Reimplemented from Hadroniser.

5.6.1.3 bool Jetset7Hadroniser::Hadronise (**Event** * ev_) [virtual]

Launches the hadroniser on the full event information

Parameters

in,out	*ev_*	The event to hadronise

Returns

A boolean stating whether or not the hadronisation occured successfully

Reimplemented from Hadroniser.

5.6.1.4 static float Jetset7Hadroniser::luchge (int pdgid_) [inline], [static], [private]

Give the charge for a parton/particle

Parameters

in	*pdgid_*	PDG id of the parton/particle

5.6.1.5 static void Jetset7Hadroniser::luexec ( ) [inline], [static], [private]

Administrate the fragmentation and decay chain. It may be called several times, but only entries which have not yet been treated (more precisely, have $1 \leq KS \leq 10$) can be affected by further calls. This may apply if more jets/particles have been added by the user, or if particles previously considered stable are now allowed to decay. The actions that will be taken during a LUEXEC call can be tailored extensively via the LUDAT1 - LUDAT3 commonblocks, in particular by setting the MSTJ values suitably.

5.6.1.6 static void Jetset7Hadroniser::lugive (const std::string & line_) [inline], [static], [private]

Set the value of any variable residing in the commmonblocks LUJETS, LUDAT1, LUDAT2, LUDAT3, LUDAT4, or LUDATR. This is done in a more controlled fashion than by directly including the commonblocks in the user program, in that array bounds are checked and the old and new values for the variable changed is written to the output for reference.

Parameters

in	*line_*	The line to be parsed and fed to the Jetset instance

5.6.1.7 static void Jetset7Hadroniser::lujoin (int njoin_, int ijoin_[2]) [inline], [static], [private]

Connect a number of previously defined partons into a string configuration.

Initially the partons must be given with status codes (KS = K(I,1)) 1, 2 or 3.

Afterwards the partons all have status code 3, i.e. are given with full colour flow information.

Compared to the normal way of defining a parton system, the partons need therefore not appear in the same sequence in the event record as they are assumed to do along the string.

It is also possible to call LUSHOW for all or some of the entries making up the string formed by *lujoin*.

Parameters

in	*njoin_*	Number of particles to be joined by one string
in	*ijoin_*	List of particles to join in the colour flow. An one-dimensional array, of size at least *njoin_*. The *njoin_* first numbers are the positions of the partons that are to be joined, given in the order the partons are assumed to appear along the string. If the system consists entirely of gluons, the string is closed by connecting back the last to the first entry.

Note

Only one string (i.e. one colour singlet) may be defined per call, but one is at liberty to use any number of *lujoin* calls for a given event. The program will check that the parton configuration specified makes sense, and not take any action unless it is. Note, however, that an initially sensible parton configuration may become nonsensical, if only some of the partons are reconnected, while the others are left unchanged.

5.6.1.8 static void Jetset7Hadroniser::lulist (int mlist_) [inline], [static], [private]

The *mlist_* parameter can take these values :

- 0 : writes a header with program version number and last date of change; is mostly for internal use.
- 1 : gives a simple list of current event record, in an 80 column format suitable for viewing directly on the computer terminal. For each entry, the following information is given:
  - the entry number I,
  - the parton/particle name (see below),
  - the status code KS (K(I,1)),
  - the flavour code KF (K(I,2)),
  - the line number of the mother (K(I,3)), and
  - the three-momentum, energy and mass (P(I,1) - P(I,5)).

  If MSTU(3) is nonzero, lines immediately after the event record proper are also listed. A final line contains information on total charge, momentum, energy and invariant mass. The particle name is given by a call to the routine LUNAME. For an entry which has decayed/fragmented (KS = 11 - 20),

this particle name is given within parantheses. Similarly, a documentation line (KS = 21 - 30) has the name enclosed in expression signs (!...!) and an event/jet axis information line the name within inequality signs (<...>). If the last character of the name is a ?, it is a signal that the complete name has been truncated to fit in, and can therefore not be trusted; this is very rare. For partons which have been arranged along strings (KS = 1, 2, 11 or 12), the end of the parton name column contains information about the colour string arrangement:

  - a A for the first entry of a string,
  - an I for all intermediate ones, and
  - a V for the final one (a poor man's vertical rendering of the doublesided arrow <-—>).

  It is possible to insert lines just consisting of sequences of ====== to separate different sections of the event record, see MSTU(70) - MSTU(80).

- 2 : gives a more extensive list of the current event record, in a 132 column format, suitable for printers or workstations. For each entry, the following information is given:

  - the entry number I,
  - the parton/particle name (with padding as described for *mlist_* = 1),
  - the status code KS (K(I,1)),
  - the flavour code KF (K(I,2)),
  - the line number of the mother (K(I,3)),
  - the decay product/colour flow pointers (K(I,4), K(I,5)), and
  - the three-momentum, energy and mass (P(I,1) - P(I,5)).

  If MSTU(3) is nonzero, lines immediately after the event record proper are also listed. A final line contains information on total charge, momentum, energy and invariant mass. Lines with only ====== may be inserted as for MLIST(1).

- 3 : gives the same basic listing as = 2, but with an additional line for each entry containing information on production vertex position and time (V(I,1) - V(I,4)) and, for unstable particles, invariant lifetime (V(I,5)).

- 11 : provides a simple list of all parton/particle codes defined in the program, with KF code and corresponding particle name. The list is grouped by particle kind, and only within each group in ascending order.

- 12 : provides a list of all parton/particle and decay data used in the program. Each parton/particle code is represented by one line containing:

  - KF flavour code,
  - KC compressed code,
  - particle name,
  - antiparticle name (where appropriate),
  - electrical and colour charge (stored in KCHG),
  - mass,
  - resonance width and maximum broadening,
  - average invariant lifetime (in PMAS) and whether the particle is considered stable or not (in MDCY).

  Immediately after a particle, each decay channel gets one line, containing:

  - decay channel number (IDC read from MDCY),
  - on/off switch for the channel,
  - matrix element type (MDME),
  - branching ratio (BRAT), and

– decay products (KFDP).

The MSTU(14) flag can be used to set the maximum flavour for which particles are listed, with the default (= 0) corresponding to separately defined ones (KC > 100 if KF > 0). In order to keep the size down, decay modes of heavy hadrons collectively defined are never listed; these have KC codes 84 - 88, where the relevant information may be found.

- 13 : gives a list of current parameter values for MSTU, PARU, MSTJ and PARJ, and the first 200 entries of PARF. This is useful to keep check of which default values were changed in a given run.

Parameters

in	*mlist_*	Determines what is to be listed (see detailed description)

5.6.1.9 static std::string Jetset7Hadroniser::luname (int pdgid_) [inline], [static], [private]

Give the parton/particle name (as a character string).

Parameters

in	*pdgid_*	PDG id of the parton/particle

5.6.1.10 static double Jetset7Hadroniser::ulmass (int pdgid_) [inline], [static], [private]

Gives the mass for a parton/particle

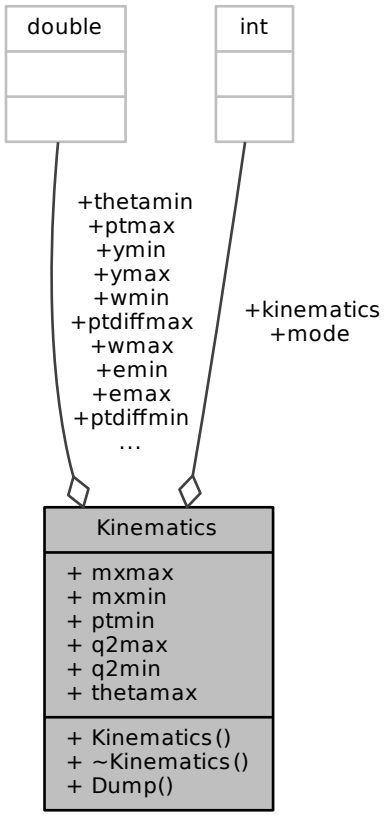
Parameters

in	*pdgid_*	PDG id of the parton/particle

5.7 Kinematics Class Reference

List of kinematic cuts to apply on the central and outgoing phase space.

Collaboration diagram for Kinematics:



Public Member Functions

- void Dump ()

  Dumps all the parameters used in this process cross-section computation / events generation.

Data Fields

- double emax

  Maximal energy of the central two-photons system.
- double emin

  Minimal energy of the central two-photons system.
- int kinematics

  Type of kinematics to consider for the phase space.
- int mode

  Sets of cuts to apply on the final phase space.
- double mxmax

  Maximal mass (in GeV/c ^{2}) of the outgoing proton remnant(s)
- double mxmin

  Minimal mass (in GeV/c ^{2}) of the outgoing proton remnant(s)

- double **ptdiffmax**
- double **ptdiffmin**
- double ptmax

  Maximal transverse momentum of the single outgoing leptons.
- double ptmin

  Minimal transverse momentum of the single outgoing leptons.
- double q2max

  The maximal value of Q^2.
- double q2min

  The minimal value of Q^2.
- double thetamax

  Maximal polar ($\theta_{\max}$) angle of the outgoing leptons, expressed in degrees.
- double thetamin

  Minimal polar ($\theta_{\min}$) angle of the outgoing leptons, expressed in degrees.
- double wmax

  The maximal s on which the cross section is integrated. If negative, the maximal energy available to the system (hence, $s = (\sqrt{s})^2$) is provided.
- double wmin

  The minimal s on which the cross section is integrated.
- double **ymax**
- double **ymin**

5.7.1 Field Documentation

5.7.1.1 int Kinematics::kinematics

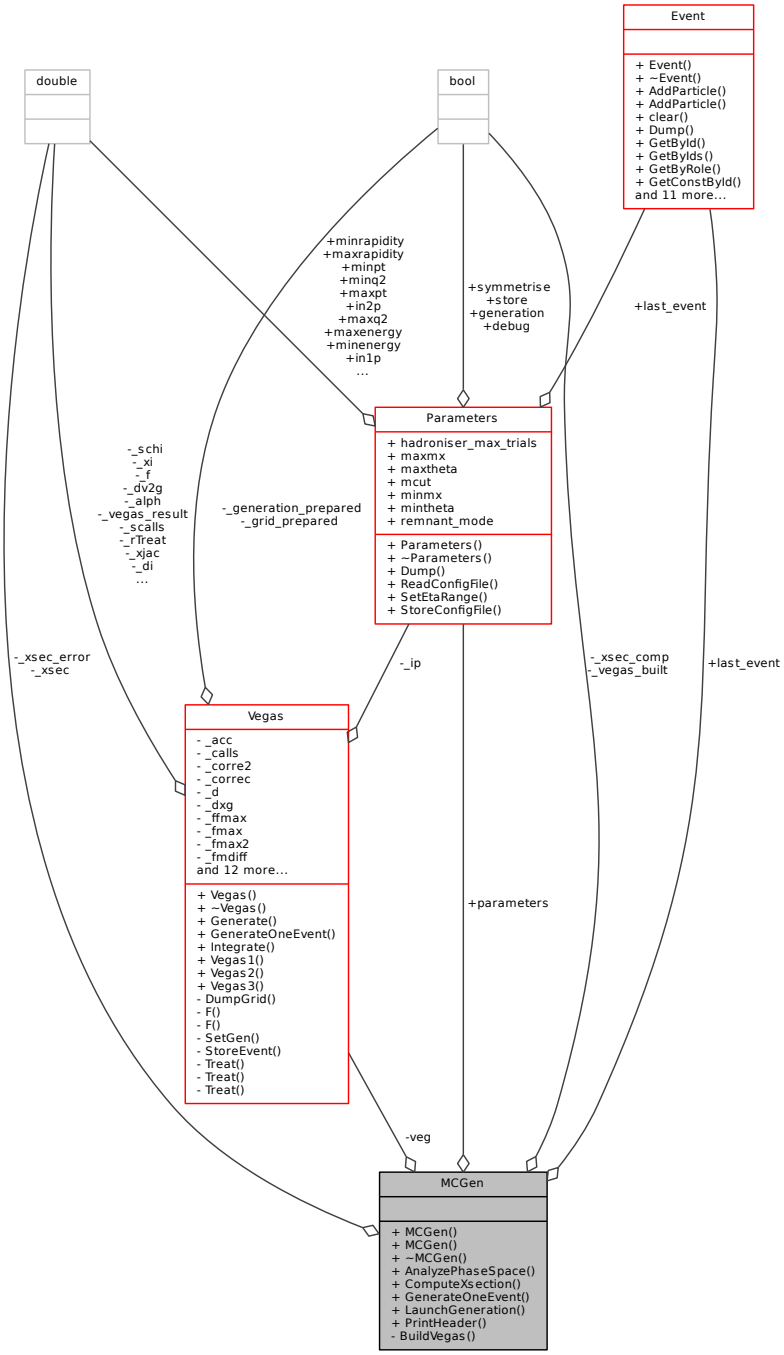
Type of kinematics to consider for the process. Can either be :

- 0 for the electron-proton elastic case
- 1 for the proton-proton elastic case
- 2 for the proton-proton single-dissociative (or inelastic) case
- 3 for the proton-proton double-dissociative case

5.8 MCGen Class Reference

Core of the Monte-Carlo generator.

Collaboration diagram for MCGen:



Public Member Functions

- MCGen ()

  Class constructor.
- MCGen (Parameters *ip_)

  Class constructor.
- void AnalyzePhaseSpace (const std::string)

Returns the set of parameters used to setup the phase space to integrate.

- void ComputeXsection (double *xsec_, double *err_)

  Compute the cross-section for the given process.
- Event * GenerateOneEvent ()
- void LaunchGeneration ()
- void PrintHeader ()

  Dumps this program's header into the standard output stream.

Data Fields

- Event * last_event

  Last event generated in this run.
- Parameters * parameters

  Physical Parameters used in the events generation and cross-section computation.

Private Member Functions

- void BuildVegas ()

  Calls the Vegas constructor (once, just before the first integration attempt)

Private Attributes

- bool _vegas_built

  Has the Vegas object already been constructed in this MCGen instance.
- double _xsec

  The cross-section computed at the last integration.
- bool _xsec_comp

  Has a first integration beed already performed ?
- double _xsec_error

  The error on the cross-section as computed at the last integration.
- Vegas * veg

  The Vegas integrator which will integrate the function.

5.8.1 Detailed Description

This object represents the core of this Monte Carlo generator, with its allowance to generate the events (using the embedded Vegas object) and to study the phase space in term of the variation of resulting cross section while scanning the various parameters (point **x** in the DIM-dimensional phase space).

The phase space is constrained using the Parameters object given as an argument to the constructor, and the differential cross-sections for each value of the array **x** are computed in the f-function defined outside (but populated inside) this object.

This f-function embeds a Process object which defines all the methods to obtain this differential cross-section as well as the in- and outgoing kinematics associated to each particle.

Author

Laurent Forthomme laurent.forthomme@uclouvain.be

Date

February 2013

5.8.2 Constructor & Destructor Documentation

5.8.2.1 MCGen::MCGen ( )

Sets the number of dimensions on which to perform the integration, according to the set of input parameters given as an argument and propagated to the whole object

5.8.2.2 MCGen::MCGen (**Parameters** * ip_)

Sets the number of dimensions on which to perform the integration, according to the set of input parameters given as an argument and propagated to the whole object

Parameters

in	*ip_*	List of input parameters defining the phase space on which to perform the integration

5.8.3 Member Function Documentation

5.8.3.1 void MCGen::AnalyzePhaseSpace (const std::string)

Returns

The Parameter object embedded in this class

5.8.3.2 void MCGen::ComputeXsection (double * xsec_, double * err_)

Computes the cross-section for the run defined by this object. This returns the cross-section as well as the absolute error computed along.

Parameters

out	*xsec_*	The computed cross-section, in pb
out	*err_*	The absolute integration error on the computed cross-section, in pb

5.8.3.3 **Event*** MCGen::GenerateOneEvent ( )

Generates one single event given the phase space computed by Vegas in the integration step

Returns

A pointer to the Event object generated in this run

5.8.3.4 void MCGen::LaunchGeneration ( )

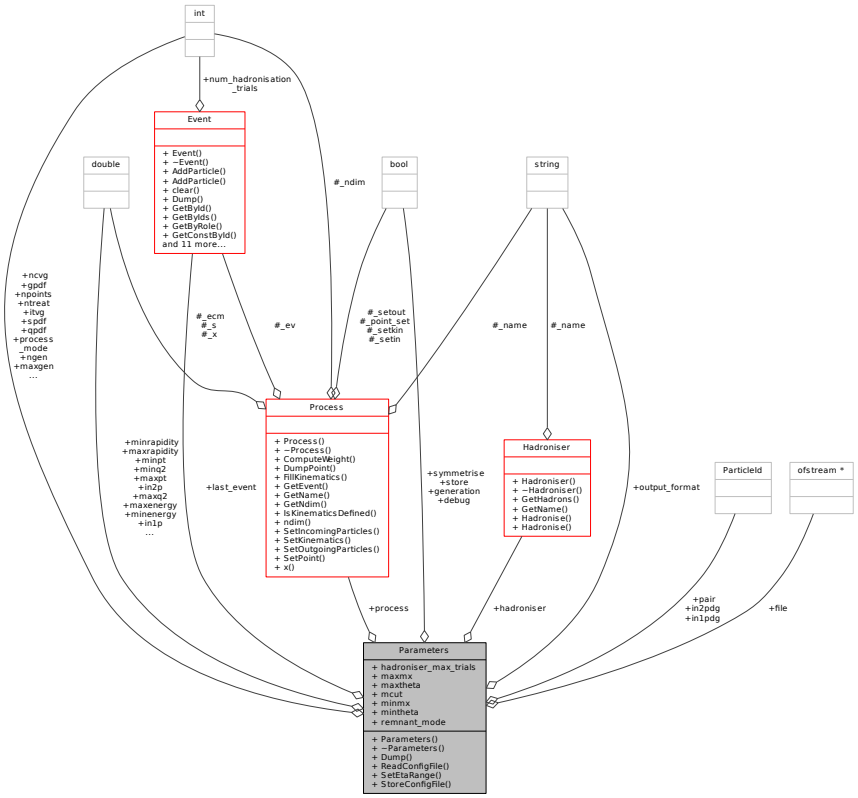
Launches the full events generation

Deprecated This method is to be suppressed since the events generation can now be launched one event at a time using the *GenerateOneEvent* method

5.9 Parameters Class Reference

List of parameters used to start and run the simulation job.

Collaboration diagram for Parameters:



Public Member Functions

- void Dump ()

  Dumps the input parameters in the console.
- bool ReadConfigFile (std::string inFile_)

  Reads content from config file to load the variables.
- void SetEtaRange (double etamin_, double etamax_)

  Sets the pseudo-rapidity range for the produced leptons.
- bool StoreConfigFile (std::string outFile_)

  Stores the full run configuration to an external config file.

Data Fields

- bool debug

  Do we need control plots all along the process?
- std::ofstream * file

  The file in which to store the events generation's output.
- bool generation

  Are we generating events ? (true) or are we only computing the cross-section ? (false)
- int gpdf

  PDFLIB group to use.
- Hadroniser * hadroniser

  Hadronisation algorithm to use for the proton(s) remnants fragmentation.

- int hadroniser_max_trials

  Maximal number of trials for the hadronisation of the proton(s) remnants.
- double in1p

  First incoming particle's momentum (in GeV/c)
- ParticleId in1pdg

  First beam/primary particle's PDG identifier.
- double in2p

  Second incoming particle's momentum (in GeV/c)
- ParticleId in2pdg

  Second beam/primary particle's PDG identifier.
- int itvg

  Maximal number of iterations to perform by VEGAS.
- Event * last_event

  The pointer to the last event produced in this run.
- double maxenergy

  Maximal energy of the outgoing leptons.
- int maxgen

  Maximal number of events to generate in this run.
- double maxmx

  Maximal M_X of the outgoing proton remnants.
- double maxpt

  Maximal p_T of the outgoing leptons.
- double maxq2

  Maximal value of Q^2, the internal photons lines' virtuality.
- double maxrapidity

  Maximal rapidity y of the outgoing leptons.
- double maxtheta

  Maximal polar angle θ of the outgoing leptons.
- int mcut

  Set of cuts to apply on the outgoing leptons.
- double minenergy

  Minimal energy of the outgoing leptons.
- double minmx

  Minimal M_X of the outgoing proton remnants.
- double minpt

  Minimal p_T of the outgoing leptons.
- double minq2

  Minimal value of Q^2, the internal photons lines' virtuality.
- double minrapidity

  Minimal rapidity y of the outgoing leptons.
- double mintheta

  Minimal polar angle θ of the outgoing leptons.
- int **ncvg**
- int ngen

  Number of events already generated in this run.
- int npoints

  Number of points to "shoot" in each integration bin by the algorithm.
- int ntreat

  Maximal number of TREAT calls.
- std::string output_format

  Type of format the event will be stored into.

- ParticleId pair

  PDG id of the outgoing leptons.
- Process * process

  The process for which the cross-section will be computed and the events will be generated.
- int **process_mode**
- int qpdf

  Number of quarks.
- int remnant_mode

  First particle's mode.
- int spdf

  PDFLIB set to use.
- bool store

  Are the events generated in this run to be stored in the output file ?
- bool symmetrise

  Do we want the events to be symmetrised with respect to the z-axis ?

5.9.1 Detailed Description

Note

The default parameters are derived from GMUINI in LPAIR

5.9.2 Member Function Documentation

5.9.2.1 bool Parameters::ReadConfigFile (std::string inFile_)

Reads the list of parameters to be used in this cross-section computation/events generation from an external input card.

Parameters

in	*inFile_*	Name of the configuration file to load

Returns

A boolean stating whether this input configuration file is correct or not

5.9.2.2 void Parameters::SetEtaRange (double etamin_, double etamax_)

Defines the range to cover in pseudo-rapidity for the outgoing leptons produced in this process. This method converts this range into a range in θ, the polar angle.

Parameters

in	*etamin_*	The minimal value of η for the outgoing leptons
in	*etamax_*	The maximal value of η for the outgoing leptons

5.9.2.3 bool Parameters::StoreConfigFile (std::string outFile_)

Parameters

in	*outFile_*	Name of the configuration file to create

Returns

A boolean stating whether this output configuration file is correctly written or not

5.9.3 Field Documentation

5.9.3.1 bool Parameters::debug

Enables or disables the production of control plots for several kinematic quantities in this process

5.9.3.2 double Parameters::maxmx

Maximal mass of the outgoing proton remnants, M_X, in GeV/c^2.

5.9.3.3 double Parameters::maxpt

Maximal transverse momentum cut to apply on the outgoing lepton(s)

5.9.3.4 int Parameters::mcut

Set of cuts to apply on the outgoing leptons in order to restrain the available kinematic phase space :

- 0 - No cuts at all (for the total cross section)
- 1 - Vermaserens' hypothetical detector cuts : for both leptons,
  - $\frac{|p_z|}{|\mathbf{p}|} \leq 0.75$ and $p_T \geq 1$ GeV/c, or
  - $0.75 < \frac{|p_z|}{|\mathbf{p}|} \leq 0.95$ and $p_z > 1$ GeV/c,
- 2 - Cuts on both the outgoing leptons, according to the provided cuts parameters
- 3 - Cuts on at least one outgoing lepton, according to the provided cut parameters

5.9.3.5 double Parameters::minmx

Minimal mass of the outgoing proton remnants, M_X, in GeV/c^2.

5.9.3.6 double Parameters::minpt

Minimal transverse momentum cut to apply on the outgoing lepton(s)

5.9.3.7 int Parameters::ntreat

Note

Is it correctly implemented ?

5.9.3.8 ParticleId Parameters::pair

The particle code of produced leptons, as defined by the PDG convention :

- 11 - for e^+e^- pairs
- 13 - for $\mu^+\mu^-$ pairs
- 15 - for $\tau^+\tau^-$ pairs

5.9.3.9 int Parameters::remnant_mode

The first incoming particle type and kind of interaction :

- 1 - electron,
- 2 - proton elastic,
- 3 - proton inelastic without parton treatment,
- 4 - proton inelastic in parton model

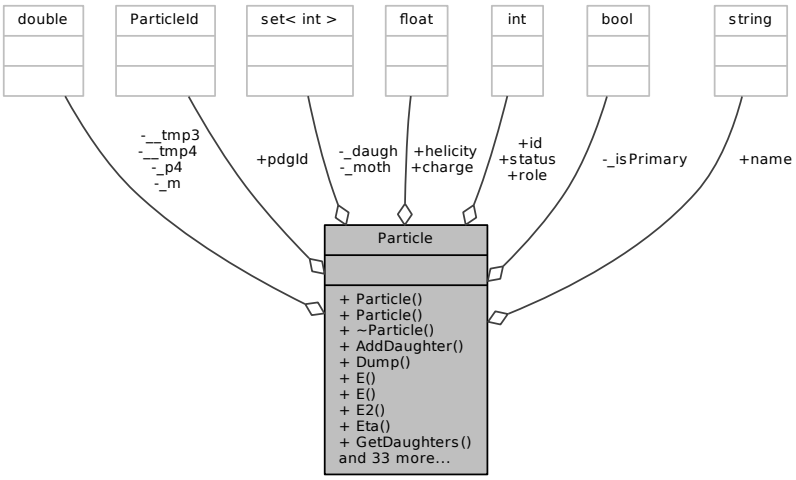
Note

Was named PMOD/EMOD in ILPAIR

5.10 Particle Class Reference

Kinematics of one particle.

Collaboration diagram for Particle:



Public Member Functions

- Particle (int role_, ParticleId pdgId_=(ParticleId) 0)

  Object constructor (providing the role of the particle in the process, and its Particle Data Group identifier)
- bool AddDaughter (Particle *part_)

  Specify a decay product for this particle.
- void Dump ()

  Dumps all the information on this particle.
- void E (double E_)

  Sets the particle's energy.
- double E () const

  Gets the particle's energy in GeV.
- double E2 () const

  Gets the particle's squared energy in GeV^2.
- double Eta ()

  Pseudo-rapidity.
- std::vector< int > GetDaughters ()

  Gets a vector containing all the daughters unique identifiers from this particle.
- std::string GetLHEline (bool revert_=false)
- ParticleIds GetMothersIds () const

  Gets the unique identifier to the mother particle from which this particle arises.
- bool Hadronise (std::string algo_)

  Hadronises the particle using Pythia.
- void **LorentzBoost** (double m_, double p_[4])

- double * LorentzBoost (double p_[3])
- double M () const

  Gets the particle's mass.
- bool M (double m_)

  Set the particle's mass in GeV/c^2.
- double M2 () const

  Gets the particle's squared mass.
- unsigned int NumDaughters ()

  Gets the number of daughter particles arising from this one.
- Particle & operator+= (const Particle &)

  Adds two particles' momenta to create a combined particle.
- Particle & operator- (const Particle &)

  Substracts two particles' momenta to extract a particle's kinematics.
- bool operator< (const Particle &rhs)

  Comparison operator to enable the sorting of particles in an event according to their unique identifier.
- bool operator< (const Particle *rhs)

  Comparison operator to enable the sorting of Particle objects' pointers in an event according to their reference's unique identifier.
- Particle & operator= (const Particle &)

  Copies all the relevant quantities from one Particle object to another.
- bool P (double px_, double py_, double pz_)

  Azimuthal angle.
- bool P (double px_, double py_, double pz_, double E_)

  Sets the 4-momentum associated to the particle.
- bool P (double p_[4])

  Sets the 4-momentum associated to the particle.
- double P (int c_) const
- double P () const

  Norm of the 3-momentum, in GeV/c.
- double * P4 ()

  Returns the particle's 4-momentum.
- std::vector< double > **P5** () const
- void **PDF2PDG** ()
- double **Phi** () const
- bool Primary () const

  Is this particle a primary particle ?
- double Pt () const

  Transverse momentum, in GeV/c.
- double Px () const

  Momentum along the x-axis in GeV/c.
- double Py () const

  Momentum along the y-axis in GeV/c.
- double Pz () const

  Momentum along the z-axis in GeV/c.
- double Rapidity ()

  Rapidity.
- void **RotateThetaPhi** (double theta_, double phi_)
- void SetMother (Particle *part_)

  Sets the mother particle (from which this particle arises)
- double **Theta** () const
- bool Valid ()

  Is this particle a valid particle which can be used for kinematic computations ?

Data Fields

- float charge

  The particle's electric charge (given as a float number, for the quarks and bound states)
- float helicity
- int id

  Unique identifier of the particle (in a Event object context)
- std::string name

  Particle's name in a human-readable format.
- ParticleId pdgId

  Particle Data Group integer identifier.
- int role

  Role in the considered process.
- int status

  Particle status.

Private Attributes

- double **___tmp3** [3]
- double **___tmp4** [4]
- ParticlesIds _daugh

  List of daughter particles.
- bool _isPrimary

  Is the particle a primary particle ?
- double _m

  Mass in GeV/c^2.
- ParticlesIds _moth

  List of mother particles.
- double _p4 [4]

5.10.1 Detailed Description

Kinematic information for one particle

5.10.2 Member Function Documentation

5.10.2.1 bool Particle::AddDaughter (**Particle** * part_)

Adds a "daughter" to this particle (one of its decay product(s))

Parameters

in	*part_*	The Particle object in which this particle will desintegrate or convert

Returns

A boolean stating if the particle has been added to the daughters list or if it was already present before

5.10.2.2 void Particle::Dump ( )

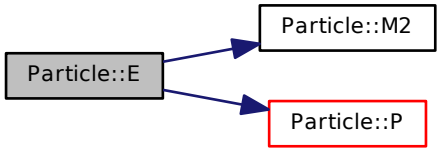
Dumps into the standard output stream all the available information on this particle

5.10.2.3 void Particle::E (double E_) [inline]

Parameters

in	*E_*	Energy, in GeV

Here is the call graph for this function:



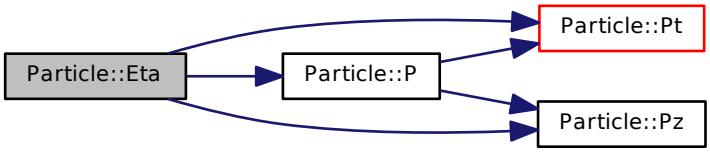
5.10.2.4 double Particle::Eta ( ) [inline]

Computes and returns η, the pseudo-rapidity of the particle

Returns

The pseudo-rapidity of the particle

Here is the call graph for this function:



5.10.2.5 std::vector<int> Particle::GetDaughters ( )

Returns

An integer vector containing all the daughters' unique identifier in the event

5.10.2.6 std::string Particle::GetLHEline (bool revert_ = false)

Returns a string containing all the particle's kinematics as expressed in the Les Houches format

Parameters

in	*revert_*	Is the event symmetric ? If set to true, the third component of the momentum is reverted.

Returns

The LHE line associated to the particle, and containing the particle's history (mother/daughters), its kinematics, and its status

5.10.2.7 ParticlesIds Particle::GetMothersIds ( ) const [inline]

Returns

An integer representing the unique identifier to the mother of this particle in the event

5.10.2.8 bool Particle::Hadronise (std::string algo_)

Hadronises the particle with Pythia, and builds the shower (list of Particle objects) embedded in this object

Parameters

in	*algo_*	Algorithm in use to hadronise the particle

Returns

A boolean stating whether or not the particle has been hadronised

5.10.2.9 double* Particle::LorentzBoost (double p_[3])

Lorentz boost from ROOT

5.10.2.10 double Particle::M ( ) const [inline]

Gets the particle's mass in GeV/c^2.

Returns

The particle's mass

5.10.2.11 bool Particle::M (double m_)

Set the mass of the particle in GeV/c^2 according to a value given as an argument. This method ensures that the kinematics is properly set (the mass is set according to the energy and the momentum in priority)

Parameters

*m_*	The mass in GeV/c^2 to set

Returns

A boolean stating whether or not the mass was correctly set

5.10.2.12 bool Particle::P (double px_, double py_, double pz_) [inline]

Computes and returns ϕ, the azimuthal angle of the particle in the transverse plane

Returns

The azimuthal angle of the particle Sets the 3-momentum associated to the particle

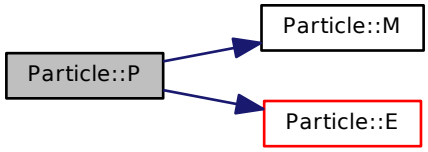
Parameters

in	*px_*	Momentum along the x-axis, in GeV/c
in	*py_*	Momentum along the y-axis, in GeV/c
in	*pz_*	Momentum along the z-axis, in GeV/c

Returns

A boolean stating the validity of this particle (according to its 4-momentum norm)

Here is the call graph for this function:



5.10.2.13 bool Particle::P (double px_, double py_, double pz_, double E_) [inline]

Sets the 4-momentum associated to the particle, and computes its (invariant) mass.

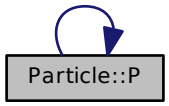
Parameters

in	*px_*	Momentum along the x-axis, in GeV/c
in	*py_*	Momentum along the y-axis, in GeV/c
in	*pz_*	Momentum along the z-axis, in GeV/c
in	*E_*	Energy, in GeV

Returns

A boolean stating the validity of the particle's kinematics

Here is the call graph for this function:



5.10.2.14 bool Particle::P (double p_[4]) [inline]

Parameters

in	*p_*	4-momentum

Returns

A boolean stating the validity of the particle's kinematics

Here is the call graph for this function:

5.10.2.15 double Particle::P (int c_) const [inline]

Get one component of the particle's 4-momentum

Parameters

in	*c_*	The component to retrieve: ▪ 0-2: $\mathbf{p} = (p_x, p_y, p_z)$ (in GeV/c) ▪ 3: E (in GeV)

Returns

The requested component of the energy-momentum for the particle

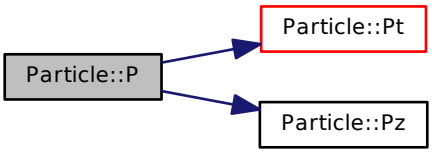
Here is the call graph for this function:

5.10.2.16 double Particle::P ( ) const [inline]

Returns

The particle's 3-momentum norm as a double precision float

Here is the call graph for this function:



5.10.2.17 double* Particle::P4 ( ) [inline]

Builds and returns the particle's 4-momentum as an array ordered as $(\mathbf{p}, E) = (p_x, p_y, p_z, E)$

Returns

The particle's 4-momentum as a 4 components double array

Here is the call graph for this function:

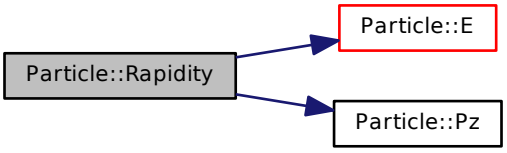
5.10.2.18 double Particle::Rapidity ( ) [inline]

Computes and returns y, the rapidity of the particle

Returns

The rapidity of the particle

Here is the call graph for this function:



5.10.2.19 void Particle::SetMother (**Particle** * part_)

Sets the "mother" of this particle (particle from which this particle is issued)

Parameters

in	*part_*	A Particle object containing all the information on the mother particle

5.10.3 Field Documentation

5.10.3.1 double Particle::_p4[4] [private]

List of components to characterise the particle's kinematics :

- 0-2: $\mathbf{p} = (p_x, p_y, p_z)$ (in GeV/c)
- 3: E (in GeV)

5.10.3.2 float Particle::helicity

Particle's helicity Float??

5.10.3.3 ParticleId Particle::pdgId

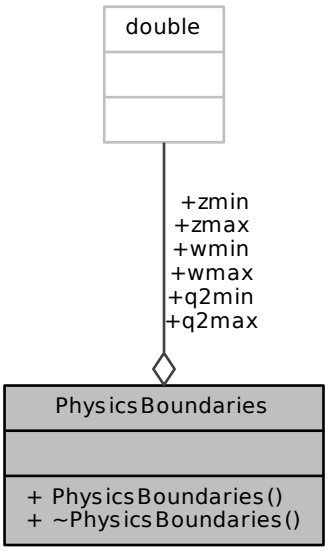
Unique identifier for a particle type. From [1] : *The Monte Carlo particle numbering scheme [...] is intended to facilitate interfacing between event generators, detector simulators, and analysis packages used in particle physics.*

5.10.3.4 int Particle::status

Codes 1-10 correspond to currently existing partons/particles, and larger codes contain partons/particles which no longer exist, or other kinds of event information

5.11 PhysicsBoundaries Class Reference

Collaboration diagram for PhysicsBoundaries:



Data Fields

- double q2max

  Maximal virtuality Q^2 of a photon in GeV ^{2}.
- double q2min

  Minimal virtuality Q^2 of a photon in GeV ^{2}.
- double wmax

  Maximal centre-of-mass energy for a γp system, in GeV.
- double wmin

  Minimal centre-of-mass energy for a γp system, in GeV.
- double zmax

  Maximal value of a generic scaling variable ζ.
- double zmin

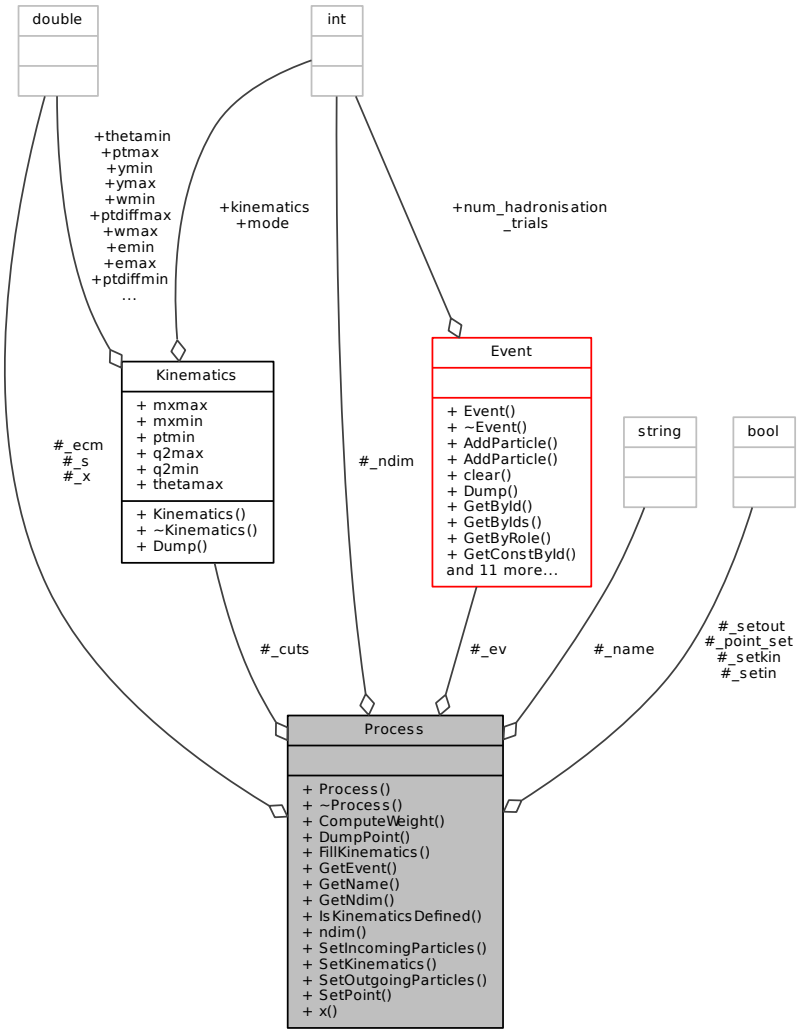
  Minimal value of a generic scaling variable ζ.

5.11.1 Detailed Description

List of physical constraints to apply on the phase space

5.12 Process Class Reference

Collaboration diagram for Process:



Public Member Functions

- virtual double ComputeWeight ()

  Returns the weight for this point in the phase-space.
- void DumpPoint ()

  Dumps the evaluated point's coordinates in the standard output stream.
- virtual void FillKinematics (bool symmetrise_=false)

  Fills the Event object with the particles' kinematics.
- Event * GetEvent ()

  Returns the event content (list of particles with an assigned role)
- std::string GetName ()

  Returns the human-readable name of the process considered.
- virtual int GetNdim (int) const

  Returns the number of dimensions on which the integration has to be performed.

- bool IsKinematicsDefined ()

  Is the system's kinematics well defined?
- unsigned int ndim () const

  Returns the number of dimensions on which the integration is performed.
- virtual bool SetIncomingParticles (Particle ip1_, Particle ip2_)

  Sets the momentum and PDG id for the incoming particles.
- virtual void SetKinematics (Kinematics cuts_)

  Sets the list of kinematic cuts to apply on the outgoing particles' final state.
- virtual bool SetOutgoingParticles (int part_, ParticleId pdgId_, int mothRole_=-1)

  Sets the PDG id for the outgoing particles.
- void SetPoint (const unsigned int ndim_, double x_[])

  Sets the phase space point to compute.
- double x (const unsigned int idx_)

  Returns the value of a component of the _ndim -dimensional point considered.

Protected Attributes

- Kinematics _cuts

  Set of cuts to apply on the final phase space.
- double _ecm

  $\sqrt{s}$, *centre of mass energy of the incoming particles' system, in* GeV
- Event * _ev

  Event object containing all the information on the in- and outgoing particles.
- std::string _name

  Name of the process (useful for logging and debugging)
- unsigned int _ndim

  Number of dimensions on which the integration has to be performed.
- bool _point_set

  Is the phase space point set ?
- double _s

  s, *squared centre of mass energy of the incoming particles' system, in* GeV^2
- bool _setin

  Are the event's incoming particles set ?
- bool _setkin

  Is the full event's kinematic set ?
- bool _setout

  Are the event's outgoing particles set ?
- double * _x

  Array of _ndim components representing the point on which the weight in the cross-section is computed.

5.12.1 Detailed Description

Class template to define any process to compute using this MC integrator/events generator

Author

Laurent Forthomme laurent.forthomme@uclouvain.be

Date

January 2014

5.12.2 Member Function Documentation

5.12.2.1 virtual void Process::FillKinematics (bool symmetrise_ = `false`) [inline], [virtual]

Fills the private Event object with all the Particle object contained in this event.

Parameters

in	*symmetrise_*	Do we have to symmetrise the event (randomise the production of the positively- and negatively-charged lepton) ?

5.12.2.2 **Event*** Process::GetEvent ( ) [inline]

Returns the complete list of Particle with their role in the process for the point considered in the phase space as an Event object.

Returns

The Event object containing all the generated Particle objects

5.12.2.3 virtual int Process::GetNdim (int) const [inline], [virtual]

Parameters

in	*process_mode-_*	Type of subprocess to consider : ▪ 1: Elastic-elastic ▪ 2: Elastic-inelastic ▪ 3: Inelastic-elastic ▪ 4: Inelastic-inelastic

Returns

Number of dimensions on which to integrate

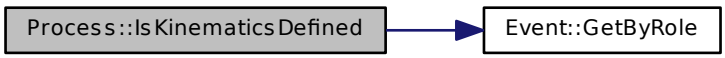
5.12.2.4 bool Process::IsKinematicsDefined ( ) [inline]

Is the system's kinematics well defined and compatible with the process ? This check is mandatory to perform the (*_ndim*)-dimensional point's cross-section computation.

Returns

A boolean stating if the input kinematics and the final states are well defined

Here is the call graph for this function:



5.12.2.5 virtual bool Process::SetIncomingParticles (**Particle** ip1_, **Particle** ip2_) [inline], [virtual]

Specifies the incoming particles' kinematics as well as their properties using two Particle objects.

Parameters

in	*ip1_*	Information on the first incoming particle
in	*ip2_*	Information on the second incoming particle

Returns

A boolean stating whether or not the incoming kinematics is properly set for this event

Here is the call graph for this function:

5.12.2.6 virtual void Process::SetKinematics (**Kinematics** cuts_) [inline], [virtual]

Parameters

in	*cuts_*	The Cuts object containing the kinematic parameters

5.12.2.7 virtual bool Process::SetOutgoingParticles (int part_, ParticleId pdgId_, int mothRole_ = -1) [inline], [virtual]

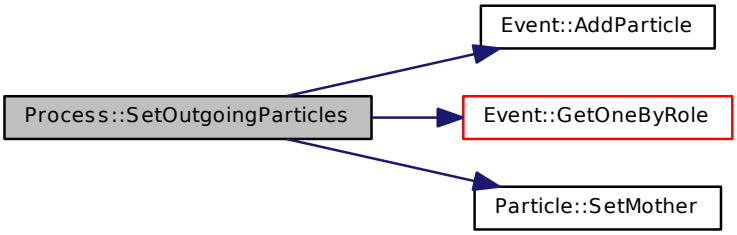
Parameters

in	*part_*	Role of the particle in the process
in	*pdgId_*	Particle ID according to the PDG convention
in	*mothRole_*	Integer role of the outgoing particle's mother

Returns

A boolean stating whether or not the outgoing kinematics is properly set for this event

Here is the call graph for this function:



5.12.2.8 void Process::SetPoint (const unsigned int ndim_, double x_[])

Sets the phase space point to compute the weight associated to it.

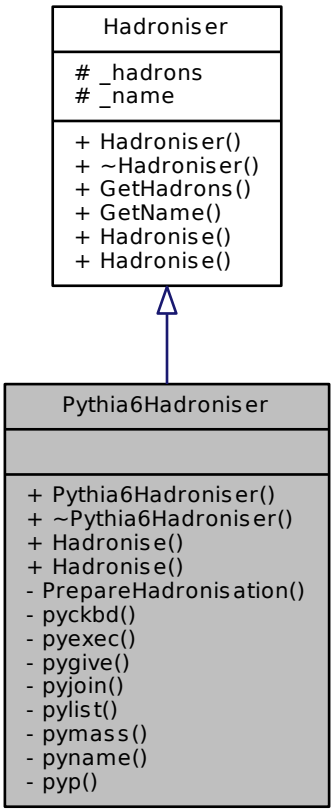
Parameters

in	*ndim_*	The number of dimensions of the point in the phase space
in	*x_[]*	The (*ndim_*)-dimensional point in the phase space on which the kinematics and the cross-section are computed

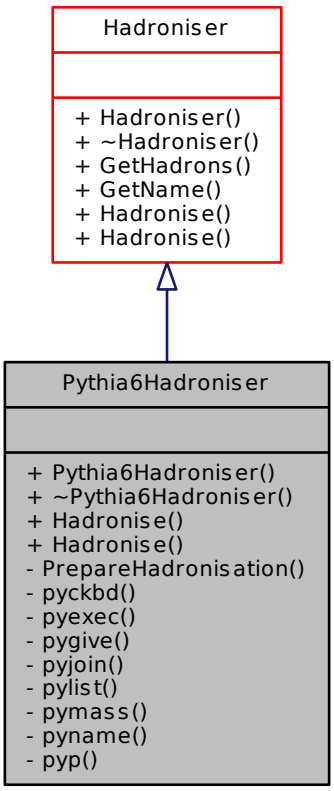
5.13 Pythia6Hadroniser Class Reference

Pythia6 hadronisation algorithm.

Inheritance diagram for Pythia6Hadroniser:



Collaboration diagram for Pythia6Hadroniser:



Public Member Functions

- Particles GetHadrons ()
- std::string GetName ()

  Returns the human-readable name of the hadroniser used.
- bool Hadronise (Particle *part_)

  Main caller to hadronise a single particle.
- bool Hadronise (Event *ev_)

  Hadronises a full event.

Protected Attributes

- Particles * _hadrons

  List of hadrons produced by this hadronisation process.
- std::string _name

  Name of the hadroniser.

Private Member Functions

- bool **PrepareHadronisation** (Event *ev_)

Static Private Member Functions

- static void **pyckbd** ()
- static void **pyexec** ()
- static void **pygive** (const std::string &line_)
- static void pyjoin (int njoin_, int ijoin_[2])

  Connect entries with colour flow information.
- static void **pylist** (int mlist_)
- static double **pymass** (int pdgid_)
- static std::string **pyname** (int pdgid_)
- static double **pyp** (int role_, int qty_)

5.13.1 Detailed Description

Full interface to the Pythia6 [3] algorithm. It can be used in a single particle decay mode as well as a full event hadronisation using the string model, as in Jetset.

5.13.2 Member Function Documentation

5.13.2.1 Particles Hadroniser::GetHadrons ( ) [inline], [inherited]

Gets the full list of hadrons (as Particle objects) produced by the hadronisation

Returns

A vector of Particle containing all the hadrons produced

5.13.2.2 bool Pythia6Hadroniser::Hadronise (**Particle** * part_) [virtual]

Parameters

in	*part_*	The Particle object which will be hadronised

Returns

A boolean stating whether or not the hadronisation occured successfully

Reimplemented from Hadroniser.

5.13.2.3 bool Pythia6Hadroniser::Hadronise (**Event** * ev_) [virtual]

Launches the hadroniser on the full event information

Parameters

in,out	*ev_*	The event to hadronise

Returns

A boolean stating whether or not the hadronisation occured successfully

Reimplemented from Hadroniser.

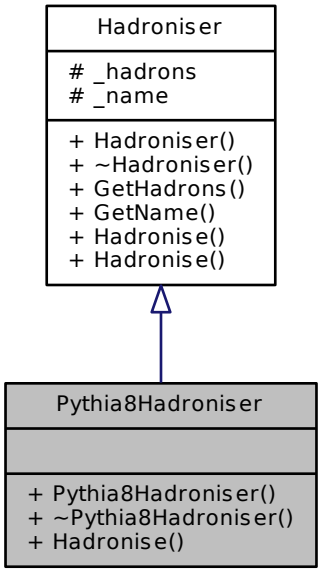
5.13.2.4 static void Pythia6Hadroniser::pyjoin (int njoin_, int ijoin_[2]) [inline], [static], [private]

Parameters

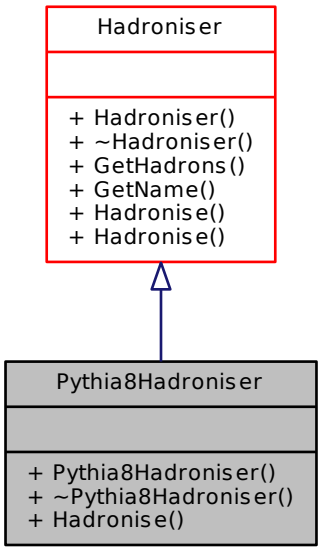
in	*njoin_*	Number of particles to join in the colour flow
in	*ijoin_*	List of particles unique identifier to join in the colour flow

5.14 Pythia8Hadroniser Class Reference

Inheritance diagram for Pythia8Hadroniser:



Collaboration diagram for Pythia8Hadroniser:



Public Member Functions

- Particles GetHadrons ()
- std::string GetName ()

  Returns the human-readable name of the hadroniser used.
- bool Hadronise (Event *ev_)

  Hadronises a full event.
- virtual bool Hadronise (Particle *part_)

  Main caller to hadronise a single particle.

Protected Attributes

- Particles * _hadrons

  List of hadrons produced by this hadronisation process.
- std::string _name

  Name of the hadroniser.

5.14.1 Member Function Documentation

5.14.1.1 Particles Hadroniser::GetHadrons ( ) [inline], [inherited]

Gets the full list of hadrons (as Particle objects) produced by the hadronisation

Returns

A vector of Particle containing all the hadrons produced

5.14.1.2 bool Pythia8Hadroniser::Hadronise (**Event** * ev_) [virtual]

Launches the hadroniser on the full event information

Parameters

in,out	*ev_*	The event to hadronise

Returns

A boolean stating whether or not the hadronisation occured successfully

Reimplemented from Hadroniser.

5.14.1.3 virtual bool Hadroniser::Hadronise (**Particle** * part_) [inline], [virtual], [inherited]

Parameters

in	*part_*	The Particle object which will be hadronised

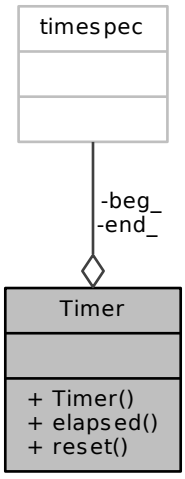
Returns

A boolean stating whether or not the hadronisation occured successfully

Reimplemented in Pythia6Hadroniser, and Jetset7Hadroniser.

5.15 Timer Class Reference

Collaboration diagram for Timer:



Public Member Functions

- double elapsed ()
- void reset ()

  Resets the clock counter.

Private Attributes

- timespec beg_

  Timestamp marking the beginning of the counter.

- timespec end_

  Timestamp marking the end of the counter.

5.15.1 Detailed Description

An object which enables to extract the processing time between two steps in this software's flow

5.15.2 Member Function Documentation

5.15.2.1 double Timer::elapsed ( ) [inline]

Get the time elapsed since the last *reset* call (or class construction)

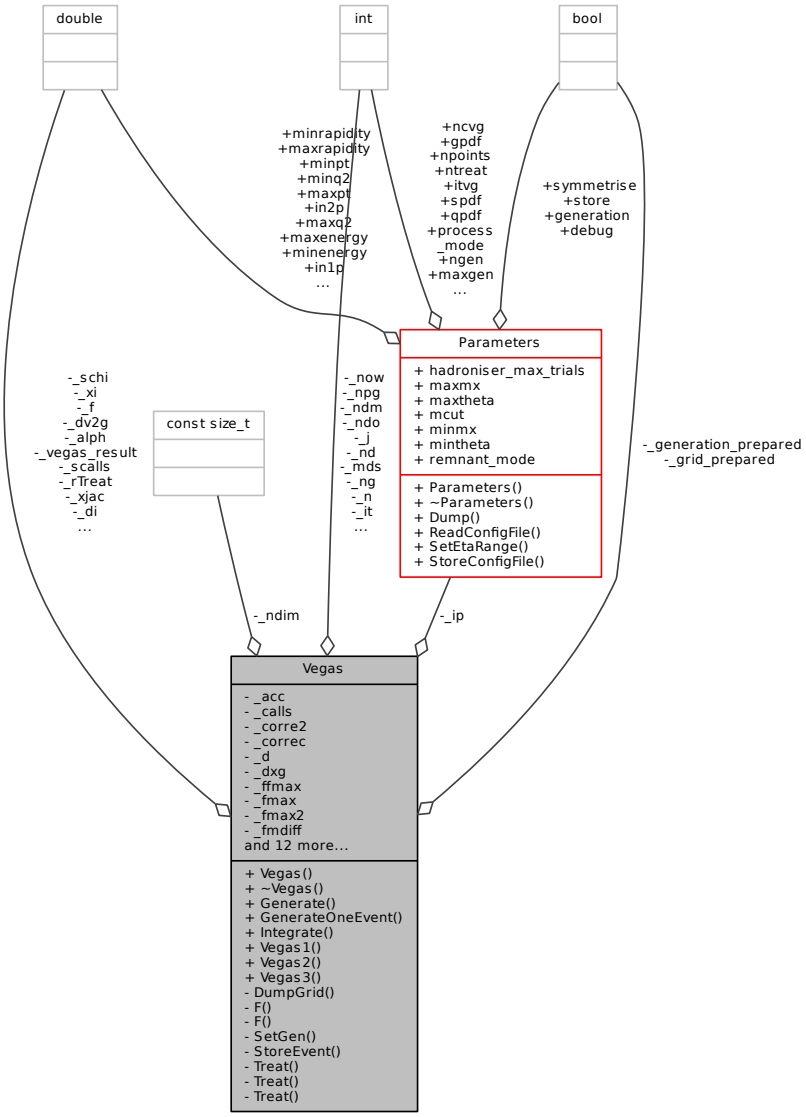
Returns

The elapsed time in seconds

5.16 Vegas Class Reference

Vegas Monte-Carlo integrator instance.

Collaboration diagram for Vegas:



Public Member Functions

- Vegas (const int dim_, double f_(double *, size_t, void *), Parameters *inParam_)
- ~Vegas ()

  Class destructor.
- void Generate ()

  Launches the generation of events.
- bool GenerateOneEvent ()

  Generates one single event according to the method defined in the Fortran 77 version of LPAIR.
- int Integrate (double *result_, double *abserr_)
- int **Vegas1** (int ncalls_=-1)
- int **Vegas2** (int ncalls_=-1)
- int **Vegas3** ()

Private Member Functions

- void DumpGrid ()
- double F (double *x_)
- double F (double *x_, Parameters *ip_)
- void SetGen ()

  Prepare the class for events generation.
- bool StoreEvent (double *x_)

  Stores the event in the output file.
- double Treat (double *x_, Parameters *ip_, bool storedbg_=false)
- double Treat (double *x_)
- double Treat (double *x_, bool storedbg_)

Private Attributes

- double **_acc**
- double **_alph**
- double **_calls**
- double **_corre2**
- double **_correc**
- double * **_d** [MAX_ND]
- double * **_di** [MAX_ND]
- double **_dv2g**
- double **_dxg**
- double(* _f)(double *x_, size_t ndim_, void *params_)

  The function which will be integrated by this Vegas instance.
- double _ffmax

  Maximal value of the function in the considered integration range.
- double * _fmax

  Maximal value of the function at one given point.
- double _fmax2

  Square of the maximal function value in the integration grid.
- double **_fmdiff**
- double **_fmold**
- bool _generation_prepared

  Flag to define whether or not the generation has been prepared using SetGen (very time-consuming operation, thus needs to be called once)
- bool _grid_prepared

  Flag to define whether or not the grid has been prepared for integration.
- Parameters * _ip

  List of parameters to specify the integration range and the physics determining the phase space.
- int **_it**
- int _j

  Selected bin at which the function will be evaluated.
- double _mbin

  Integration grid size parameter.
- int _mds

  Total number of iterations for the current Vegas instance.
- int * **_n**
- unsigned int **_nd**
- const size_t _ndim

  The number of dimensions on which to integrate the function.
- unsigned int **_ndm**

- unsigned int **_ndo**
- int **_ng**
- int * **_nm**
- int **_now**
- unsigned int **_npg**
- int _nTreatCalls
- double _rTreat
- double **_scalls**
- double **_schi**
- double **_si**
- double **_si2**
- double **_swgt**
- double **_vegas_abserr**
- double **_vegas_result**
- double _weight

  Weight of the point in the total integration.
- double * **_xi** [MAX_ND]
- double **_xjac**
- double * _xl

  Lower bounds for the points to generate.
- double **_xnd**
- double * _xu

  Upper bounds for the points to generate.

5.16.1 Detailed Description

Main occurence of the Monte-Carlo integrator[2] developed by G.P. Lepage in 1978

5.16.2 Constructor & Destructor Documentation

5.16.2.1 Vegas::Vegas (const int dim_, double f_double *, size_t, void *, **Parameters** * inParam_)

Constructs the class by booking the memory and structures for the Vegas integrator. This code is based on the Vegas Monte Carlo integration algorithm developed by P. Lepage, as documented in [2]

Parameters

in	*dim_*	The number of dimensions on which the function will be integrated
in	*f_*	The function one is required to integrate
in,out	*inParam_*	A list of parameters to define the phase space on which this integration is performed (embedded in an Parameters object)

5.16.3 Member Function Documentation

5.16.3.1 void Vegas::DumpGrid ( ) [private]

Debugging method used to dump the integration grid in the standard output stream.

5.16.3.2 double Vegas::F (double * x_) [inline], [private]

Evaluates the function to be integrated at a point *x_*, using the default Parameters object *_ip*

Parameters

in	*x_*	The point at which the function is to be evaluated

Returns

Function value at this point *x_*

5.16.3.3 double Vegas::F (double * x_, **Parameters** * ip_) [inline], [private]

Evaluates the function to be integrated at a point *x_*, given a set of Parameters *ip_*

Parameters

in	*x_*	The point at which the function is to be evaluated
in	*ip_*	A set of parameters to fully define the function

Returns

Function value at this point *x_*

5.16.3.4 void Vegas::Generate ( )

Launches the Vegas generation of events according to the provided input parameters.

5.16.3.5 bool Vegas::GenerateOneEvent ( )

Generates one event according to the grid parameters set in Vegas::SetGen

Returns

A boolean stating if the generation was successful (in term of the computed weight for the phase space point)

5.16.3.6 int Vegas::Integrate (double * result_, double * abserr_)

Vegas algorithm to perform the (_dim)-dimensional Monte Carlo integration of a given function as described in [2]

Author

Primary author : G.P. Lepage
This C++ implementation : L. Forthomme

Date

Sep 1976
Reviewed in Apr 1978
FTN5 version 21 Aug 1984
This C++ implementation is from 12 Dec 2013

Parameters

out	*result_*	The cross section as integrated by Vegas for the given phase space restrictions

out	*abserr_*	The error associated to the computed cross section

Returns

0 if the integration was performed successfully

5.16.3.7 void Vegas::SetGen ( ) [private]

Sets all the generation mode variables and align them to the integration grid set while computing the cross-section

5.16.3.8 bool Vegas::StoreEvent (double * x_) [private]

Stores the event characterized by its _ndim-dimensional point in the phase space to the output file

Parameters

in	*x_*	The *_ndim-dimensional* point in the phase space defining the unique event to store

Returns

A boolean stating whether or not the event could be saved

5.16.3.9 double Vegas::Treat (double * x_, **Parameters** * ip_, bool storedbg_ = false) [private]

Transforms the function to integrate into a numerically stable function where poles are tamed.

Parameters

in	*x_*	The *_ndim* -dimensional point at which the stabilised function is to be evaluated
in	*ip_*	The physics parameters to apply to the function to evaluate
in	*storedbg_*	A debugging flag to set whether or not the internal variables of this method need to be stored for further processing

Returns

Tamed function value at this point *x_*

5.16.3.10 double Vegas::Treat (double * x_) [inline], [private]

Evaluates the smoothed version of the function to be integrated at a point *x_*, using the default Parameters object *_ip*

Parameters

in	*x_*	The point at which the tamed function is to be evaluated

Returns

Tamed function value at this point *x_*

Here is the call graph for this function:

5.16.3.11 double Vegas::Treat (double * x_, bool storedbg_) [inline], [private]

Evaluates the smoothed version of the function to be integrated at a point *x_*

Parameters

in	*x_*	The point at which the tamed function is to be evaluated

Returns

Tamed function value at this point *x_*

Here is the call graph for this function:

5.16.4 Field Documentation

5.16.4.1 double(* Vegas::_f)(double *x_, size_t ndim_, void *params_) [private]

Parameters

*x_*	The point at which this function is evaluated
*ndim_*	The number of degrees of freedom this function has
*params_*	A "_void_-ified" Parameters object to define the boundaries of the phase space (physics constraints)

5.16.4.2 int Vegas::_nTreatCalls [private]

Has the Treat function already been called once ?

5.16.4.3 double Vegas::_rTreat [private]

$r = \text{ndo}^{\text{ndim}}$ value of the Treat function

References

[1] J. Beringer et al. Review of Particle Physics (RPP). *Phys.Rev.*, D86:010001, 2012. 40

[2] G. Peter Lepage. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, 27(2):192 – 203, 1978. 1, 58, 59

[3] Torbjorn Sjostrand, Stephen Mrenna, and Peter Z. Skands. PYTHIA 6.4 Physics and Manual. *JHEP*, 0605:026, 2006. 50

[4] J.A.M. Vermaseren. Two-photon processes at very high energies. *Nuclear Physics B*, 229(2):347 – 371, 1983. 1

Index