

LPAIR++
0.2

Generated by Doxygen 1.8.6

Mon Nov 10 2014 12:39:56

Contents

1 Principles	1
2 Todo List	2
3 Deprecated List	2
4 Hierarchical Index	2
4.1 Class Hierarchy	2
5 Data Structure Index	2
5.1 Data Structures	3
6 Data Structure Documentation	3
6.1 Event Class Reference	3
6.1.1 Detailed Description	5
6.1.2 Member Function Documentation	5
6.1.3 Field Documentation	9
6.2 GamGamLL Class Reference	9
6.2.1 Detailed Description	17
6.2.2 Constructor & Destructor Documentation	18
6.2.3 Member Function Documentation	18
6.2.4 Field Documentation	24
6.3 GamGamWW Class Reference	24
6.3.1 Member Function Documentation	27
6.4 GamPomVMLL Class Reference	29
6.4.1 Member Function Documentation	35
6.4.2 Field Documentation	43
6.5 Hadroniser Class Reference	47
6.5.1 Detailed Description	48
6.5.2 Member Function Documentation	48
6.6 Herwig6Hadroniser Class Reference	49
6.6.1 Member Function Documentation	50
6.7 Jetset7Hadroniser Class Reference	51
6.7.1 Member Function Documentation	54
6.8 Kinematics Class Reference	57
6.8.1 Field Documentation	59
6.9 MCGen Class Reference	59
6.9.1 Detailed Description	61
6.9.2 Constructor & Destructor Documentation	62
6.9.3 Member Function Documentation	62

6.10 Parameters Class Reference	62
6.10.1 Detailed Description	65
6.10.2 Member Function Documentation	65
6.10.3 Field Documentation	65
6.11 Particle Class Reference	67
6.11.1 Detailed Description	69
6.11.2 Member Function Documentation	69
6.11.3 Field Documentation	75
6.12 PhysicsBoundaries Class Reference	76
6.12.1 Detailed Description	76
6.13 PPtoLL Class Reference	76
6.13.1 Member Function Documentation	79
6.14 Process Class Reference	82
6.14.1 Detailed Description	84
6.14.2 Member Function Documentation	84
6.15 Pythia6Hadroniser Class Reference	88
6.15.1 Detailed Description	90
6.15.2 Member Function Documentation	90
6.16 Pythia8Hadroniser Class Reference	91
6.16.1 Member Function Documentation	92
6.17 Timer Class Reference	94
6.17.1 Detailed Description	95
6.17.2 Member Function Documentation	95
6.18 Vegas Class Reference	95
6.18.1 Detailed Description	98
6.18.2 Constructor & Destructor Documentation	98
6.18.3 Member Function Documentation	98
6.18.4 Field Documentation	101
Bibliography	103
Index	104

1 Principles



This Monte Carlo generator, based on the LPAIR code developed in the early 1990s by J. Vermaseren *et al*[7], allows to compute the cross-section and to generate events for the $\gamma\gamma \rightarrow \ell^+\ell^-$ 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[3].

2 Todo List

Global `GamGamLL::GamGamLL (int nOpt_=0)`

Figure out how this *nOpt_* parameter is affecting the final cross-section computation and events generation

3 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

4 Hierarchical Index

4.1 Class Hierarchy

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

Event	3
Hadroniser	47
Herwig6Hadroniser	49
Jetset7Hadroniser	51
Pythia6Hadroniser	88
Pythia8Hadroniser	91
Kinematics	57
MCGen	59
Parameters	62
Particle	67
PhysicsBoundaries	76
Process	82
GamGamLL	9
GamGamWW	24
GamPomVMML	29
PPtoLL	76
Timer	94
Vegas	95

5 Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

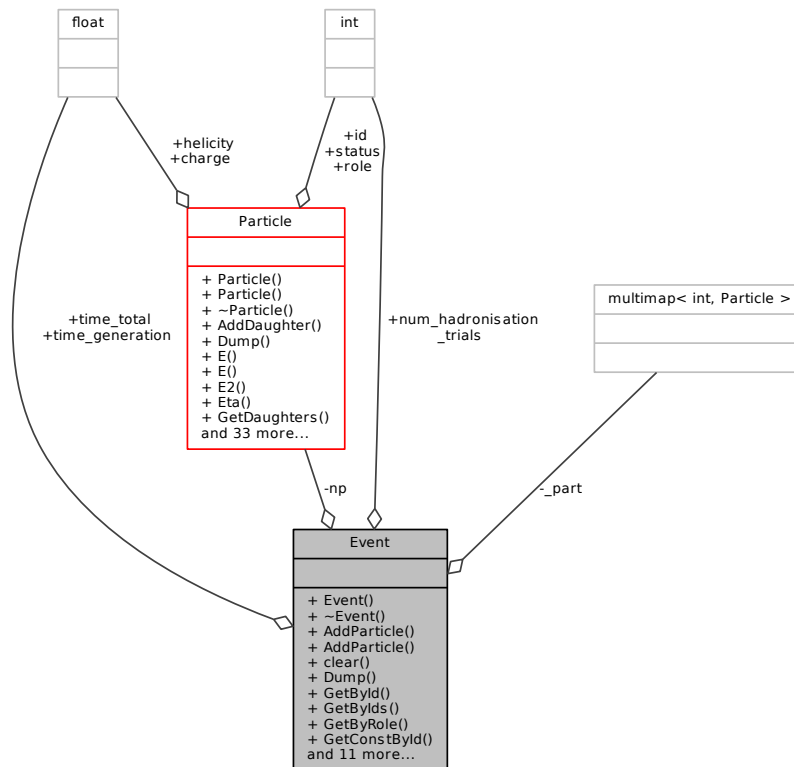
Event	
Kinematic information on the particles in the event	3
GamGamLL	
Computes the matrix element for a CE $\gamma\gamma \rightarrow \ell^+\ell^-$ process	9
GamGamWW	
Computes the matrix element for a CE $\gamma\gamma \rightarrow W^+W^-$ process	24
GamPomVMLL	
Computes the matrix element for a CE $\gamma\mathbb{P} \rightarrow \rho, \omega, \phi, J/\psi, \Upsilon, \dots \rightarrow \ell^+\ell^-$ process	29
Hadroniser	47
Herwig6Hadroniser	
Herwig6 hadronisation algorithm	49
Jetset7Hadroniser	
Jetset7 hadronisation algorithm	51
Kinematics	
List of kinematic cuts to apply on the central and outgoing phase space	57
MCGen	
Core of the Monte-Carlo generator	59
Parameters	
List of parameters used to start and run the simulation job	62
Particle	
Kinematics of one particle	67
PhysicsBoundaries	76
PPtoLL	
Computes the matrix element for a CE $\gamma\gamma \rightarrow \ell^+\ell^-$ process using k_T -factorization approach	76
Process	82
Pythia6Hadroniser	
Pythia6 hadronisation algorithm	88
Pythia8Hadroniser	91
Timer	94
Vegas	
Vegas Monte-Carlo integrator instance	95

6 Data Structure Documentation

6.1 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 * GetByld (int id_)`
Gets one particle by its unique identifier in the event.
- `ParticlesRef GetBylds (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 GetConstByld (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`

6.1.1 Detailed Description

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

6.1.2 Member Function Documentation

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

Sets the information on one particle in the process

Parameters

<code>in</code>	<code>part_</code>	The Particle object to insert or modify in the event
<code>in</code>	<code>replace_</code>	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

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

Parameters

in	<i>role_</i>	The role the particle will play in the process
in	<i>replace_</i>	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

6.1.2.3 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	<i>stable_</i>	Do we only show the stable particles in this event ?
-----------	----------------	--

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

Returns the pointer to the [Particle](#) object corresponding to a unique identifier in the event

Parameters

in	<i>id_</i>	The unique identifier to this particle in the event
-----------	------------	---

Returns

A pointer to the requested [Particle](#) object

6.1.2.5 **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	<i>ids_</i>	The unique identifiers to the particles to be selected in the event
-----------	-------------	---

Returns

A vector of pointers to the requested [Particle](#) objects

6.1.2.6 **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	<i>role_</i>	The role the particles have to play in the process
-----------	--------------	--

Returns

A vector of pointers to the requested [Particle](#) objects

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

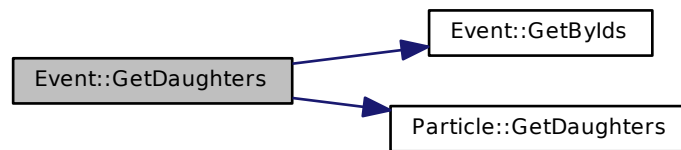
Parameters

in	<i>part_</i>	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:

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

Returns the pointer to the mother particle of any given [Particle](#) object in this event

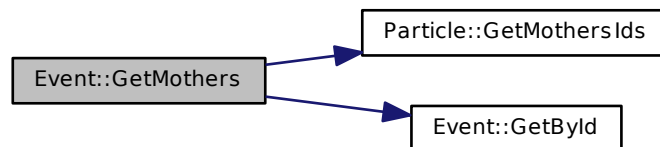
Parameters

in	<i>part_</i>	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:

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

Returns the first [Particle](#) object in the particles list whose role corresponds to the given argument

Parameters

in	<i>role_</i>	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:

6.1.2.10 `ParticlesRef Event::GetParticles ()`

Returns

A vector containing all the pointers to the [Particle](#) objects contained in the event

6.1.2.11 `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

6.1.2.12 `ParticlesRef Event::GetStableParticles ()`

Returns

A vector containing all the pointers to the stable [Particle](#) objects contained in the event

6.1.2.13 `int Event::NumParticles () const [inline]`

Returns

The number of particles in the event, as an integer

6.1.2.14 `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	<i>weight_</i>	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

<code>in</code>	<code>weight_</code>	The weight of the event
-----------------	----------------------	-------------------------

6.1.3 Field Documentation

6.1.3.1 ParticlesMap Event::_part [private]

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

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

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

6.1.3.3 float Event::time_generation

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

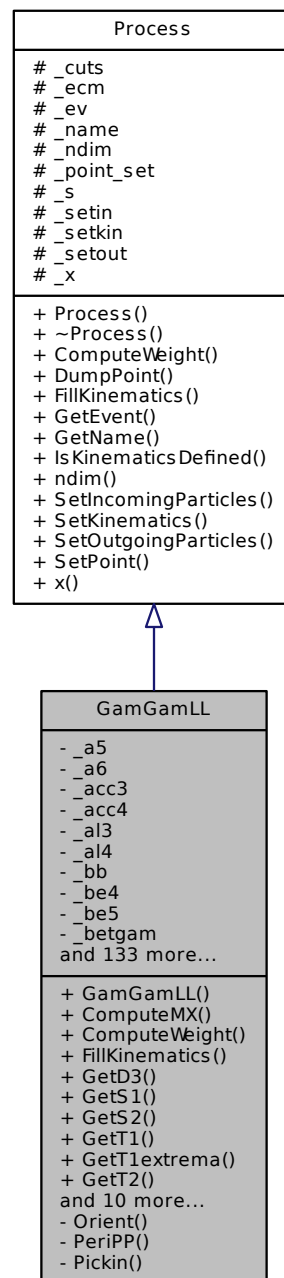
6.1.3.4 float Event::time_total

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

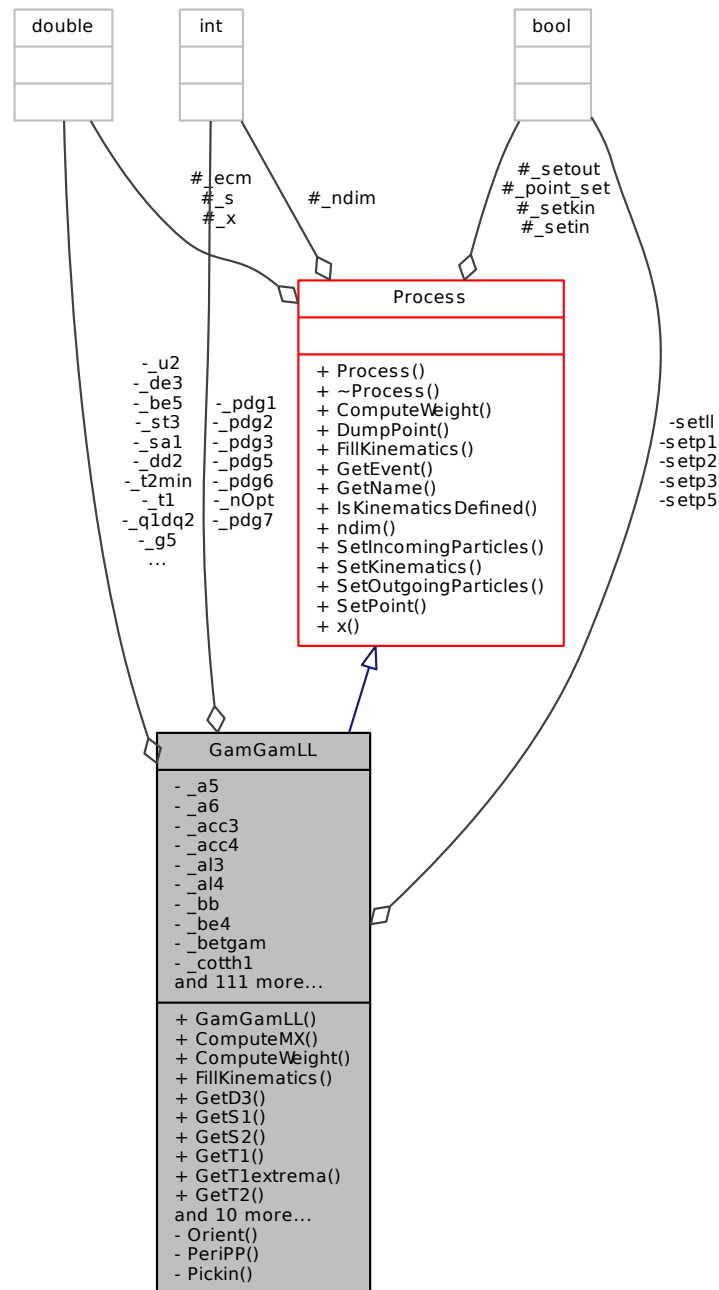
6.2 GamGamLL Class Reference

Computes the matrix element for a CE $\gamma\gamma \rightarrow \ell^+\ell^-$ process.

Inheritance diagram for GamGamLL:



Collaboration diagram for GamGamLL:



Public Member Functions

- **GamGamLL** (int nOpt_=0)
Class constructor.
- double **ComputeMX** (double x_, double outmass_, double *dw_)
Computes the outgoing proton remnant mass.
- double **ComputeWeight** ()

- *Computes the process' weight for the given point.*
- void **DumpPoint** ()
 - *Dumps the evaluated point's coordinates in the standard output stream.*
- void **FillKinematics** (bool)
 - *Fills the **Event** object with the particles' kinematics.*
- double **GetD3** ()
- **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.*
- double **GetS1** ()
- double **GetS2** ()
- double **GetT1** ()
- void **GetT1extrema** (double &t1min_, double &t1max_)
- double **GetT2** ()
- void **GetT2extrema** (double &t2min_, double &t2max_)
- double **GetU1** ()
- double **GetU2** ()
- double **GetV1** ()
- double **GetV2** ()
- bool **IsKinematicsDefined** ()
 - *Is the system's kinematics well defined?*
- unsigned int **ndim** () const
 - *Returns the number of dimensions on which the integration is performed.*
- void **PrepareHadronisation** (**Particle** *part_)
- bool **SetIncomingParticles** (**Particle**, **Particle**)
 - *Sets the momentum and PDG id for the incoming particles.*
- void **SetKinematics** (**Kinematics**)
 - *Sets the list of kinematic cuts to apply on the outgoing particles' final state.*
- bool **SetOutgoingParticles** (int, ParticleId, int)
 - *Sets the PDG id for the outgoing particles.*
- void **SetPoint** (const unsigned int ndim_, double x_[])
 - *Sets the phase space point to compute.*
- void **StoreEvent** (std::ofstream *, double)
- 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.

Private Member Functions

- bool [Orient](#) ()
Energies/momenta computation for the various particles, in the CM system.
- double [PeriPP](#) (int, int)
Computes the matrix element squared for the requested process.
- bool [Pickin](#) ()

Private Attributes

- double [_a5](#)
- double [_a6](#)
- double [_acc3](#)
- double [_acc4](#)
- double [_al3](#)
- double [_al4](#)
- double [_bb](#)
- double [_be4](#)
- double [_be5](#)
- double [_betgam](#)
 $\beta\gamma$ factor of the centre-of-mass system, used in the computation of the inverse boost for the outgoing leptons
- double [_cotth1](#)
- double [_cotth2](#)
- double [_cp3](#)
 $\cos\phi_3$ of the first outgoing proton-like particle
- double [_cp5](#)
 $\cos\phi_5$ of the second outgoing proton-like particle
- double [_cp6](#)
 $\cos\phi_6$ of the first outgoing lepton
- double [_cp7](#)
 $\cos\phi_7$ of the second outgoing lepton
- double [_ct3](#)
 $\cos\theta_3$ of the first outgoing proton-like particle
- double [_ct4](#)
 $\cos\theta_4$ of the two-photons centre-of-mass system
- double [_ct5](#)
 $\cos\theta_5$ of the second outgoing proton-like particle
- double [_ct6](#)
 $\cos\theta_6$ of the first outgoing lepton
- double [_ct7](#)
 $\cos\theta_7$ of the second outgoing lepton
- double [_ctcm6](#)

- $\cos\theta_6^{\text{CM}}$, production angle of the first outgoing lepton, computed in the centre-of-mass system.
- double **_d1dq**
- double **_d1dq2**
- double **_d3**
- double **_dd1**
- double **_dd2**
- double **_dd3**
- double **_dd4**
- $\delta_5 = m_4^2 - t_1$ as defined in Vermaseren's paper [7] for the full definition of this quantity
- double **_dd5**
- double **_de3**
- double **_de5**
- double **_delta**
- double **_dj**
- double **_dw31**
- double **_dw52**
- double **_e6lab**
- E_6^{lab} , energy of the first outgoing lepton, computed in the lab frame
- double **_e7lab**
- E_7^{lab} , energy of the second outgoing lepton, computed in the lab frame
- double **_ec4**
- E_4 , energy of the two-photon central system
- double **_eg1**
- Energy of the first central photon of momentum t_1 .
- double **_eg2**
- Energy of the second central photon of momentum t_2 .
- double **_el6**
- E_6 , energy of the first outgoing lepton
- double **_el7**
- E_7 , energy of the second outgoing lepton
- double **_ep1**
- E_1 , energy of the first proton-like incoming particle
- double **_ep2**
- E_2 , energy of the second proton-like incoming particle
- double **_ep3**
- E_3 , energy of the first proton-like outgoing particle
- double **_ep5**
- E_5 , energy of the second proton-like outgoing particle
- double **_epsi**
- double **_etot**
- Total energy provided by the two incoming proton-like particles.
- double **_g4**
- double **_g5**
- double **_g6**
- double **_gamma**
- γ factor of the centre-of-mass system, used in the computation of the inverse boost for the outgoing leptons
- double **_gram**
- double **_mc4**
- m_4 , mass of the two-photon central system
- double **_ml6**
- m_6 , mass of the first outgoing lepton
- double **_ml7**

- m_7 , mass of the second outgoing lepton
- double [_mp1](#)
- m_1 , mass of the first proton-like incoming particle
- double [_mp2](#)
- m_2 , mass of the second proton-like incoming particle
- double [_mp3](#)
- m_3 , mass of the first proton-like outgoing particle
- double [_mp5](#)
- m_5 , mass of the second proton-like outgoing particle
- int [_nOpt](#)
- double [_p](#)
- double [_p12](#)
- $$p_{12} = \frac{1}{2} (s - m_{p_1}^2 - m_{p_2}^2)$$
- double [_p13](#)
- $$p_{13} = -\frac{1}{2} (t_1 - m_{p_1}^2 - m_{p_3}^2)$$
- double [_p14](#)
- double [_p15](#)
- double [_p1k2](#)
- double [_p23](#)
- double [_p24](#)
- double [_p25](#)
- double [_p2k1](#)
- double [_p34](#)
- double [_p35](#)
- double [_p3_g1](#) [3]
- $3\text{-momentum of the second central photon of momentum } t_1$
- double [_p3_g2](#) [3]
- $3\text{-momentum of the second central photon of momentum } t_2$
- double [_p45](#)
- double [_p_p3](#)
- double [_p_p4](#)
- double [_p_p5](#)
- double [_pc4](#)
- $|\mathbf{p}_4|$, 3-momentum norm of the two-photon central system
- ParticleId [_pdg1](#)
- $PDG \text{ identifier of the first proton-like incoming particle.}$
- ParticleId [_pdg2](#)
- $PDG \text{ identifier of the second proton-like incoming particle.}$
- ParticleId [_pdg3](#)
- $PDG \text{ identifier of the first proton-like outgoing particle.}$
- ParticleId [_pdg5](#)
- $PDG \text{ identifier of the second proton-like outgoing particle.}$
- ParticleId [_pdg6](#)
- $PDG \text{ identifier of the first outgoing lepton.}$
- ParticleId [_pdg7](#)
- $PDG \text{ identifier of the second outgoing lepton.}$
- double [_pl6](#)
- $|\mathbf{p}_6|$, 3-momentum norm of the first outgoing lepton
- double [_pl7](#)
- $|\mathbf{p}_7|$, 3-momentum norm of the second outgoing lepton
- double [_pp1](#)
- $|\mathbf{p}_1|$, 3-momentum norm of the first proton-like incoming particle

- double [_pp2](#)
 $|\mathbf{p}_2|$, 3-momentum norm of the second proton-like incoming particle
- double [_pp3](#)
 $|\mathbf{p}_3|$, 3-momentum norm of the first proton-like outgoing particle
- double [_pp5](#)
 $|\mathbf{p}_5|$, 3-momentum norm of the second proton-like outgoing particle
- double [_pt_l6](#)
 $p_{T,6}$, transverse momentum of the first outgoing lepton
- double [_pt_l7](#)
 $p_{T,7}$, transverse momentum of the second outgoing lepton
- double [_ptot](#)
Total momentum provided by the two incoming proton-like particles (along the z -axis)
- double [_q1dq](#)
- double [_q1dq2](#)
- double [_q2max](#)
Maximal Q^2 exchange.
- double [_q2min](#)
Minimal Q^2 exchange.
- double [_qp2max](#)
- double [_qp2min](#)
- double [_s1](#)
- double [_s2](#)
- double [_sa1](#)
- double [_sa2](#)
- double [_sl1](#)
- double [_sp3](#)
 $\sin \phi_3$ of the first outgoing proton-like particle
- double [_sp5](#)
 $\sin \phi_5$ of the second outgoing proton-like particle
- double [_sp6](#)
 $\sin \phi_6$ of the first outgoing lepton
- double [_sp7](#)
 $\sin \phi_7$ of the second outgoing lepton
- double [_st3](#)
 $\sin \theta_3$ of the first outgoing proton-like particle
- double [_st4](#)
 $\sin \theta_4$ of the two-photons centre-of-mass system
- double [_st5](#)
 $\sin \theta_5$ of the second outgoing proton-like particle
- double [_st6](#)
 $\sin \theta_6$ of the first outgoing lepton
- double [_st7](#)
 $\sin \theta_7$ of the second outgoing lepton
- double [_stcm6](#)
 $\sin \theta_6^{\text{CM}}$, production angle of the first outgoing lepton, computed in the centre-of-mass system.
- double [_t1](#)
- double [_t1max](#)
- double [_t1min](#)
- double [_t2](#)
- double [_t2max](#)
- double [_t2min](#)
- double [_tau](#)

- $\delta_6 = m_4^2 - m_5^2$ as defined in Vermaseren's paper [7] for the full definition of this quantity
- double [_u1](#)
- double [_u2](#)
- double [_v1](#)
- double [_v2](#)
- double [_w1](#)
- m_1^2 , squared mass of the first proton-like incoming particle
- double [_w12](#)
- $\delta_2 = m_1^2 - m_2^2$ as defined in Vermaseren's paper [7] for the full definition of this quantity
- double [_w2](#)
- m_2^2 , squared mass of the second proton-like incoming particle
- double [_w3](#)
- m_3^2 , squared mass of the first proton-like outgoing particle
- double [_w31](#)
- $\delta_1 = m_3^2 - m_1^2$ as defined in Vermaseren's paper [7] for the full definition of this quantity
- double [_w4](#)
- m_4^2 , squared mass of the two-photon central system
- double [_w5](#)
- m_5^2 , squared mass of the second proton-like outgoing particle
- double [_w52](#)
- $\delta_4 = m_5^2 - m_2^2$ as defined in Vermaseren's paper [7] for the full definition of this quantity
- double [_w6](#)
- m_6^2 , squared mass of the first outgoing lepton
- double [_w7](#)
- m_7^2 , squared mass of the second outgoing lepton
- bool [setll](#)
- *Is the outgoing leptons' state set ?*
- bool [setp1](#)
- *Is the first incoming proton-like particle's kinematic set ?*
- bool [setp2](#)
- *Is the second incoming proton-like particle's kinematic set ?*
- bool [setp3](#)
- *Is the first outgoing proton-like particle's kinematic set ?*
- bool [setp5](#)
- *Is the second outgoing proton-like particle's kinematic set ?*

6.2.1 Detailed Description

Full class of methods and objects to compute the full analytic matrix element [7] for the $\gamma\gamma \rightarrow \ell^+\ell^-$ process according to a set of kinematic constraints provided for the incoming and outgoing particles (the [Kinematics](#) object). The particle roles in this process are defined as following :

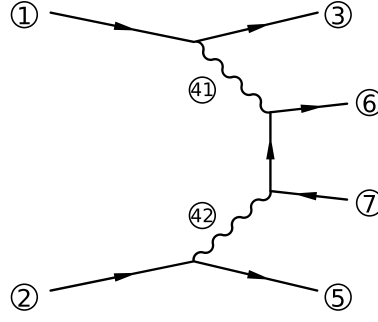


Figure 1: Detailed particle roles in the two-photon process as defined by the *GamGamLL* object. The incoming protons/electrons are denoted by a role 1, and 2, as the outgoing protons/protons remnants/ electrons carry the indices 3 and 5. The two outgoing leptons have the roles 6 and 7, while the lepton/antilepton distinction is done randomly (thus, the arrow convention is irrelevant here).

The f function created by this [Process](#) child has its `_ndim` -dimensional coordinates mapped as :

- $0 = t_1$, first incoming photon's virtuality
- $1 = t_2$, second incoming photon's virtuality
- $2 = s_2$ mapping
- $3 = yy4 = \cos(\pi x_3)$ definition
- $4 = w_4$, the two-photon system's invariant mass
- $5 = xx6 = \frac{1}{2} (1 - \cos \theta_6^{\text{CM}})$ definition (3D rotation of the first outgoing lepton with respect to the two-photon centre-of-mass system). If the `nm_` optimisation flag is set this angle coefficient value becomes

$$\frac{1}{2} \left(\frac{a_{\text{map}}}{b_{\text{map}}} \frac{\beta - 1}{\beta + 1} + 1 \right)$$

with $a_{\text{map}} = \frac{1}{2} (w_4 - t_1 - t_2)$, $b_{\text{map}} = \frac{1}{2} \sqrt{((w_4 - t_1 - t_2)^2 - 4t_1t_2) \left(1 - 4\frac{w_6}{w_4}\right)}$, and $\beta = \left(\frac{a_{\text{map}} + b_{\text{map}}}{a_{\text{map}} - b_{\text{map}}}\right)^{2x_5 - 1}$

and the `_dj` element is scaled by a factor $\frac{1}{2} \frac{(a_{\text{map}}^2 - b_{\text{map}}^2 \cos^2 \theta_6^{\text{CM}})}{a_{\text{map}} b_{\text{map}}} \log \left(\frac{a_{\text{map}} + b_{\text{map}}}{a_{\text{map}} - b_{\text{map}}} \right)$

- $6 = phicm6$, or ϕ_6^{CM} the rotation angle of the dilepton system in the centre-of-mass system
- $7 = x_q, w_X$ mappings, as used in the single- and double-dissociative cases only

6.2.2 Constructor & Destructor Documentation

6.2.2.1 GamGamLL::GamGamLL (int nOpt_ = 0)

Sets the mandatory parameters used in the methods computing the kinematics and the cross-section for the $\gamma\gamma \rightarrow \ell^+\ell^-$ process

Parameters

in	nOpt_	Optimisation???
----	-------	-----------------

Todo Figure out how this `nOpt_` parameter is affecting the final cross-section computation and events generation

6.2.3 Member Function Documentation

6.2.3.1 double GamGamLL::ComputeMX (double x_, double outmass_, double * dw_)

Computes the mass of the outgoing proton remnant if any

Parameters

in	x_{-}	A random number (between 0 and 1)
in	$outmass_{-}$	The maximal outgoing particles' invariant mass
out	dw_{-}	The size of the integration bin

Returns

The mass of the outgoing proton remnant

6.2.3.2 double GamGamLL::ComputeWeight () [virtual]

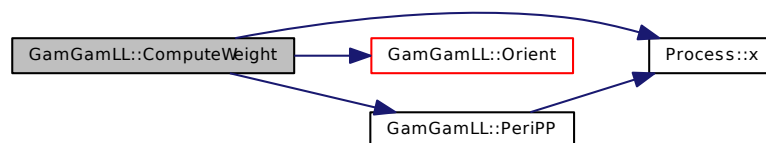
Computes the cross-section for the $\gamma\gamma \rightarrow \ell^+\ell^-$ process with the given kinematics

Returns

$d\sigma(\mathbf{x})(\gamma\gamma \rightarrow \ell^+\ell^-)$, the differential cross-section for the given point in the phase space.

Reimplemented from [Process](#).

Here is the call graph for this function:



6.2.3.3 void GamGamLL::FillKinematics (bool symmetrise_) [virtual]

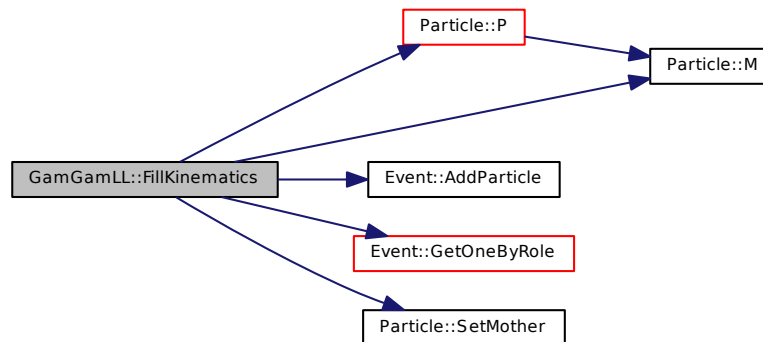
Fills the private [Event](#) object with all the [Particle](#) object contained in this event.

Parameters

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

Reimplemented from [Process](#).

Here is the call graph for this function:



6.2.3.4 **Event*** Process::GetEvent () [inline], [inherited]

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

6.2.3.5 double GamGamLL::GetT1 () [inline]

Returns the value for the first photon virtuality

Returns

t_1 , the first photon virtuality

6.2.3.6 void GamGamLL::GetT1extrema (double & t1min_, double & t1max_) [inline]

Returns the two limit values for the first photon virtuality

Parameters

out	<i>t1min_</i>	The minimal value for t_1
out	<i>t1max_</i>	The maximal value for t_1

6.2.3.7 double GamGamLL::GetT2 () [inline]

Returns the value for the second photon virtuality

Returns

t_2 , the second photon virtuality

6.2.3.8 void GamGamLL::GetT2extrema (double & t2min_, double & t2max_) [inline]

Returns the two limit values for the second photon virtuality

Parameters

out	<i>t2min_</i>	The minimal value for t_2
out	<i>t2max_</i>	The maximal value for t_2

6.2.3.9 bool Process::IsKinematicsDefined () [inline], [inherited]

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:



6.2.3.10 bool GamGamLL::Orient () [private]

Calculates energies and momenta of the 1st, 2nd (resp. the "proton-like" and the "electron-like" incoming particles), 3rd (the "proton-like" outgoing particle), 4th (the two-photons central system) and 5th (the "electron-like" outgoing particle) particles in the overall centre-of-mass frame.

Here is the call graph for this function:



6.2.3.11 double GamGamLL::PeriPP (int nup_, int ndown_) [private]

Contains the expression of the matrix element squared for the $\gamma\gamma \rightarrow \ell^+\ell^-$ process. It returns the value of the convolution of the form factor or structure functions with the central two-photons matrix element squared.

Returns

The full matrix element for the two-photon production of a pair of spin $-\frac{1}{2}$ -point particles. It is noted as

$$M = \frac{1}{4bt_1t_2} \sum_{i=1}^2 \sum_{j=1}^2 u_i v_j t_{ij} = \frac{1}{4} \frac{u_1 v_1 t_{11} + u_2 v_1 t_{21} + u_1 v_2 t_{12} + u_2 v_2 t_{22}}{t_1 t_2 b}$$

where $b = _bb$ is defined in *ComputeWeight* as :

$$b = t_1 t_2 + \left(w_{\gamma\gamma} \sin^2 \theta_6^{\text{CM}} + 4m_\ell \cos^2 \theta_6^{\text{CM}} \right) p_g^2$$

Here is the call graph for this function:



6.2.3.12 `bool GamGamLL::Pickin () [private]`

Describes the kinematics of the process $p_1 + p_2 \rightarrow p_3 + p_4 + p_5$ in terms of Lorentz-invariant variables. These variables (along with others) will then be feeded into the *PeriPP* method (thus are essential for the evaluation of the full matrix element).

Here is the call graph for this function:



6.2.3.13 `void GamGamLL::PrepareHadronisation (Particle * part_)`

Sets all the kinematic variables for the outgoing proton remnants in order to be able to hadronise them afterwards

Parameters

<code>in</code>	<code>part_</code>	Particle to "prepare" for the hadronisation to be performed
-----------------	--------------------	---

6.2.3.14 `bool GamGamLL::SetIncomingParticles (Particle ip1_, Particle ip2_) [virtual]`

Specifies the incoming particles' kinematics as well as their properties using two [Particle](#) objects.

Parameters

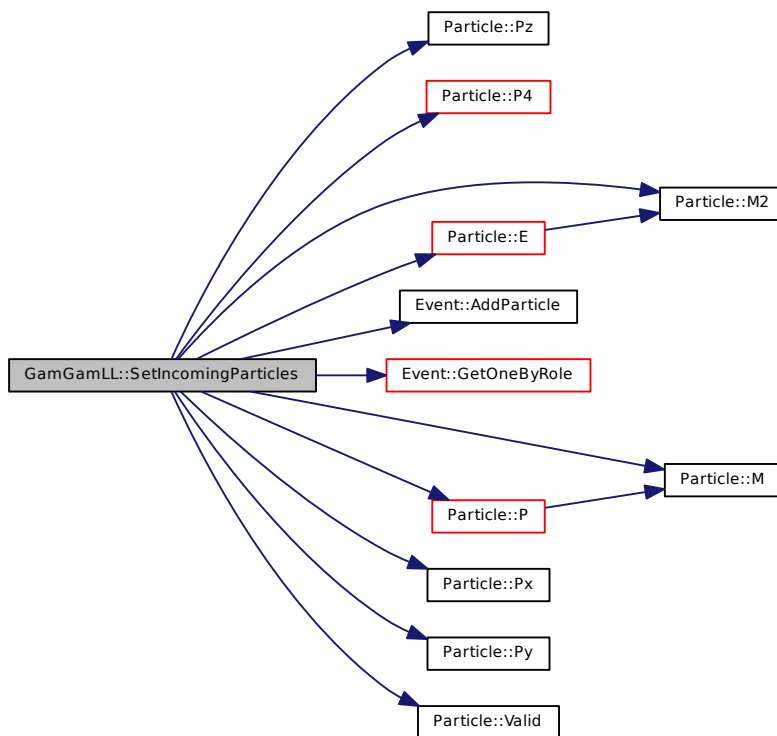
<code>in</code>	<code>ip1_</code>	Information on the first incoming particle
<code>in</code>	<code>ip2_</code>	Information on the second incoming particle

Returns

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

Reimplemented from [Process](#).

Here is the call graph for this function:



6.2.3.15 `void GamGamLL::SetKinematics (Kinematics cuts_) [virtual]`

Parameters

in	<code>cuts_</code>	The Cuts object containing the kinematic parameters
-----------	--------------------	---

Reimplemented from [Process](#).

6.2.3.16 `bool GamGamLL::SetOutgoingParticles (int part_, ParticleId pdgId_, int mothRole_) [virtual]`

Parameters

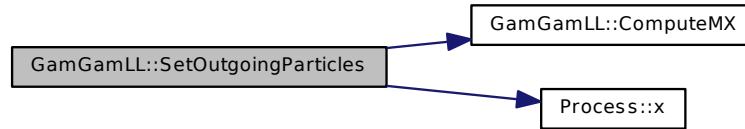
in	<code>part_</code>	Role of the particle in the process
in	<code>pdgId_</code>	Particle ID according to the PDG convention
in	<code>mothRole_</code>	Integer role of the outgoing particle's mother

Returns

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

Reimplemented from [Process](#).

Here is the call graph for this function:



6.2.3.17 `void Process::SetPoint (const unsigned int ndim_, double x_[]) [inherited]`

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

Parameters

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

6.2.4 Field Documentation

6.2.4.1 `double GamGamLL::_delta [private]`

Invariant used to tame divergences in the matrix element computation. It is defined as

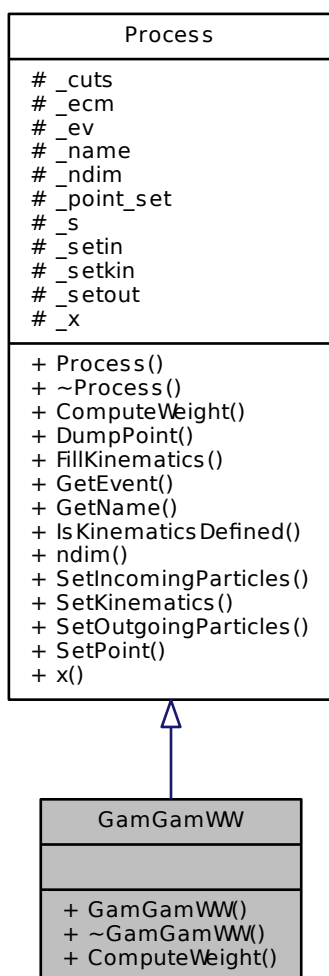
$$\Delta = (p_1 \cdot p_2)(q_1 \cdot q_2) - (p_1 \cdot q_2)(p_2 \cdot q_1)$$

with p_i, q_i the 4-momenta associated to the incoming proton-like particle and to the photon emitted from it.

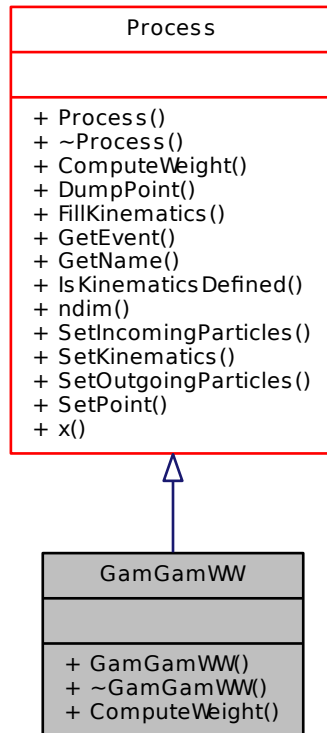
6.3 GamGamWW Class Reference

Computes the matrix element for a CE $\gamma\gamma \rightarrow W^+W^-$ process.

Inheritance diagram for GamGamWW:



Collaboration diagram for GamGamWW:



Public Member Functions

- `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.
- `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 GeV²
- 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.

6.3.1 Member Function Documentation

6.3.1.1 virtual void Process::FillKinematics (bool symmetrise_ = false) [inline], [virtual], [inherited]

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) ?
--------------------	-----------------------------	--

Reimplemented in [GamGamLL](#), and [GamPomVMLL](#).

6.3.1.2 **Event*** Process::GetEvent () [inline], [inherited]

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

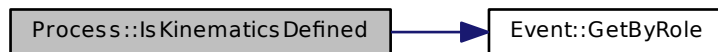
6.3.1.3 `bool Process::IsKinematicsDefined () [inline], [inherited]`

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:



6.3.1.4 `virtual bool Process::SetIncomingParticles (Particle ip1_, Particle ip2_) [inline], [virtual], [inherited]`

Specifies the incoming particles' kinematics as well as their properties using two [Particle](#) objects.

Parameters

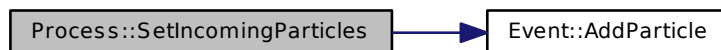
in	<i>ip1_</i>	Information on the first incoming particle
in	<i>ip2_</i>	Information on the second incoming particle

Returns

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

Reimplemented in [GamGamLL](#).

Here is the call graph for this function:



6.3.1.5 `virtual void Process::SetKinematics (Kinematics cuts_) [inline], [virtual], [inherited]`

Parameters

in	<i>cuts_</i>	The Cuts object containing the kinematic parameters
-----------	--------------	---

Reimplemented in [GamGamLL](#).

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

Parameters

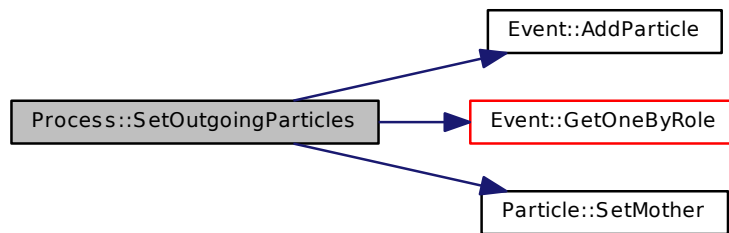
in	<i>part_</i>	Role of the particle in the process
in	<i>pdgId_</i>	Particle ID according to the PDG convention
in	<i>mothRole_</i>	Integer role of the outgoing particle's mother

Returns

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

Reimplemented in [GamGamLL](#).

Here is the call graph for this function:



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

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

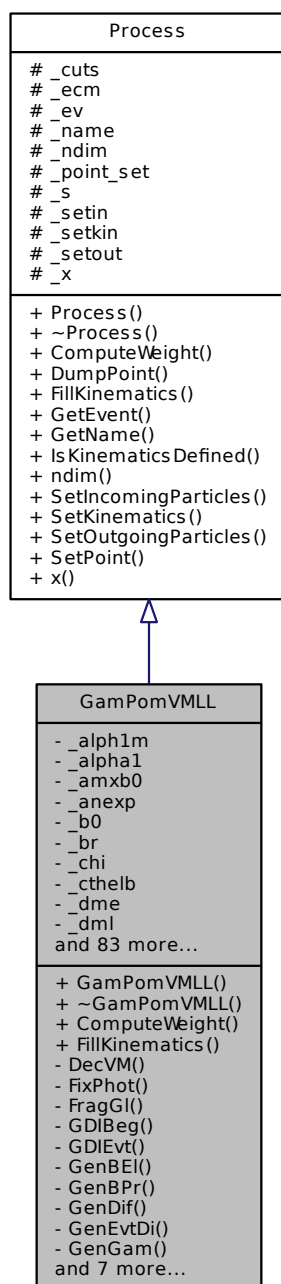
Parameters

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

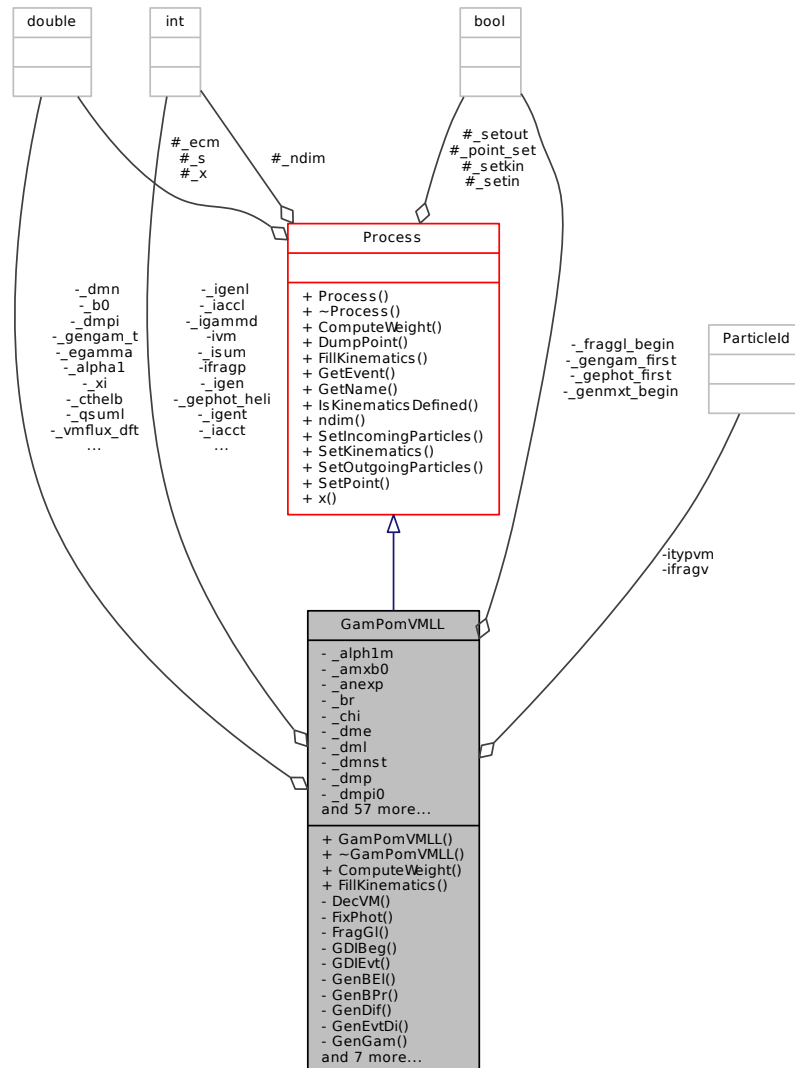
6.4 GamPomVMLL Class Reference

Computes the matrix element for a CE $\gamma\mathbb{P} \rightarrow \rho, \omega, \phi, J/\psi, \Upsilon, \dots \rightarrow \ell^+ \ell^-$ process.

Inheritance diagram for GamPomVMLL:



Collaboration diagram for GamPomVMML:



Public Member Functions

- 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.
- void [FillKinematics](#) (bool)
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.
- 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.

Private Member Functions

- void `DecVM` ()
- void `FixPhot` (`Particle` *phot_, `Particle` *ele_, double *q2_, `Particle` pel_, double egamma_)
- void `FragGI` ()
- void `GDIBeg` ()
- void `GDIEvt` ()
- void `GenBEI` ()
- void `GenBPr` ()
- void `GenDif` ()
- void `GenEvtDi` ()
- void `GenGam` ()
- double `GenMXT` (double *wght)

- void **GenPhot** ([Particle](#) *phot_, [Particle](#) *ele_, double *q2_, [Particle](#) pel_, double emin_, double q2max_)
- void **GEPhot** (double *q2_, int *heli_)
- double **OneEvent** ()
- double **PXMass** (double mmin_, double mmax_)
- void **VMFlux** ()
- double **VXMass** (double mmin_, double mmax_)

Private Attributes

- double **_alph1m**
- double **_alpha1**
Slope α' of pomeron trajectory in GeV^{-2} .
- double **_amxb0**
*Mass of diffractively dissociating hadronic system for which **_b0** was measured.*
- double **_anexp**
- double **_b0**
- double **_br**
- double **_chi**
- double **_cthelb**
- double **_dme**
- double **_dml**
- double **_dmn**
- double **_dmnst**
- double **_dmp**
- double **_dmpi**
- double **_dmpi0**
- double **_dmvm**
Mass of generated vector meson.
- double **_dsuml**
- double **_dsumt**
- double **_dwnst**
- double **_dwvm**
Width of generated vector meson.
- double **_e1**
- double **_e2**
- double **_eelmin**
- double **_egamma**
- double **_eprop**
- double **_epsilm**
Intercept of pomeron trajectory minus 1.
- double **_epsilw**
Intercept of pomeron trajectory minus 1.
- double **_event_egammin**
- int **_event_heli**
- double **_event_propmx**
- double **_event_smax**
- bool **_fraggl_begin**
- bool **_gengam_first**
- double **_gengam_t**
- double **_gengam_w2**
- double **_gengam_yhat**
- double **_genmxt_b**

- bool `__genmxt_begin`
- double `__genmxt_bmin`
- double `__genmxt_dmvp`
Mass at the proton-pomeron vertex.
- double `__genmxt_dmvv`
Mass at the vector meson-pomeron vertex.
- bool `__gephot_first`
- int `__gephot_heli`
- double `__gephot_pel` [5]
- double `__gephot_ppe` [5]
- double `__gephot_pph` [5]
- double `__gephot_ppr` [5]
- int `__iaccl`
- int `__iacct`
- int `__igammd`
- int `__igen`
- int `__igenl`
- int `__igent`
- int `__isum`
- double `__lambda`
- double `__pcm3`
CM momentum of outgoing particles.
- double `__pcmvm` [3]
- double `__photint_swei`
- double `__photint_swei2`
- double `__photint_sweil`
- double `__photint_sweil2`
- double `__photint_sweit`
- double `__photint_sweit2`
- double `__ppcms8` [1000][5]
- double `__pz1`
- double `__pz2`
- double `__q2`
Absolute of square-momentum of virtual photon.
- double `__q2max`
Maximal Q^2 of photon in GeV^2 .
- double `__q2min`
Minimal Q^2 of photon in GeV^2 .
- double `__qsuml`
- double `__qsumt`
- double `__vmflux_df`
- double `__vmflux_dfl`
- double `__vmflux_dft`
- double `__vmflux_f`
- double `__vmflux_fl`
- double `__vmflux_ft`
- double `__wb0`
CM energy of γp system at which `__b0` was measured, in GeV .
- double `__wmax`
Maximal CM energy of γp system.
- double `__wmin`
Minimal CM energy of γp system.
- double `__wsig0`

- γp CM energy at which SIGGP was measured
- double [_xi](#)
- double [_ymax](#)
 - Maximal value of scaling variable y .
- double [_ymin](#)
 - Minimal value of scaling variable y .
- double [amassv](#)
- double [deminp](#)
- int [idifv](#)
 - Index of diffractive $q\bar{q}$ states.
- int [ifragp](#)
- ParticleId [ifragv](#)
- int [ipom](#)
 - Index of pomeron photon.
- ParticleId [itypvm](#)
 - Type of vector meson to produce and its decay channel.
- int [ivm](#)
 - Index of vector meson.
- int [ivvm](#)
 - Index of virtual vector meson.

6.4.1 Member Function Documentation

6.4.1.1 void GamPomVMLL::DecVM () [private]

Let the generated vector meson decay

Author

Benno List

Date

25 Jan 1993

6.4.1.2 void GamPomVMLL::FillKinematics (bool symmetrise_) [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) ?
--------------------	-----------------------------	--

Reimplemented from [Process](#).

6.4.1.3 void GamPomVMLL::FixPhot ([Particle](#) * phot_, [Particle](#) * ele_, double * q2_, [Particle](#) pel_, double egamma_) [private]

Generate photon with fixed energy [_egamma](#), and calculate scattered electron kinematics.

Parameters

in	<i>pel_</i>	Incoming electron kinematics
in	<i>egamma_</i>	Photon energy
out	<i>phot_</i>	Photon kinematics (mass is set to $-\sqrt{Q^2}$)
out	<i>ele_</i>	Outgoing electron kinematics
out	<i>q2_</i>	Absolute Photon momentum squared (positive!)

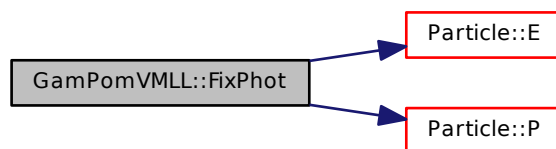
Author

Benno List

Date

28 Apr 1993

Here is the call graph for this function:



6.4.1.4 void GamPomVMLL::GDIBeg () [private]

Set up the generator for event generation

Author

Benno List

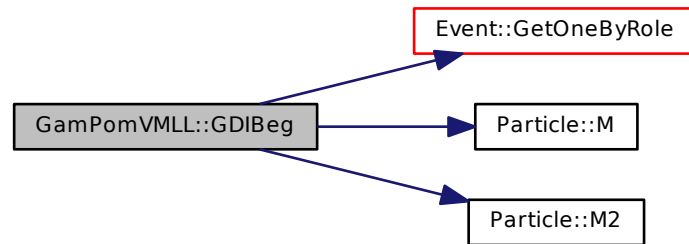
Date

31 Jan 1993 (INIGEN)
14 Jun 1995 (GDIBEG)

Note

Some of the parameters are set/calculated/changed during this step

Here is the call graph for this function:



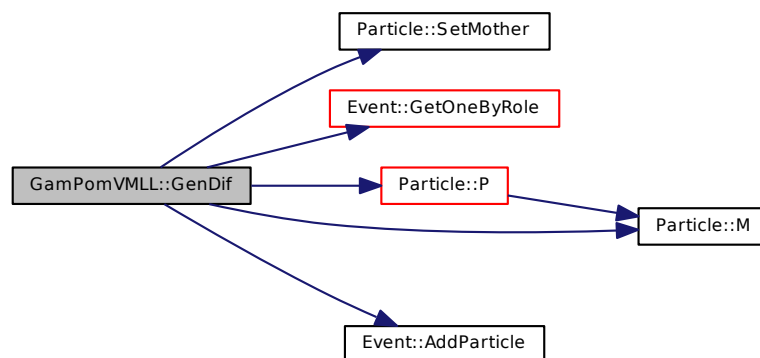
6.4.1.5 void GamPomVMLL::GenDif () [private]

Take 5-vectors of colliding γ and p and generate a diffractive state

Parameters

in	<i>tpom_</i>	Momentum transfer of the pomeron
in	<i>yhat_</i>	$\hat{y} = \sin^2(\theta^*/2)$ where θ^* is the scattering angle in the γp CMS

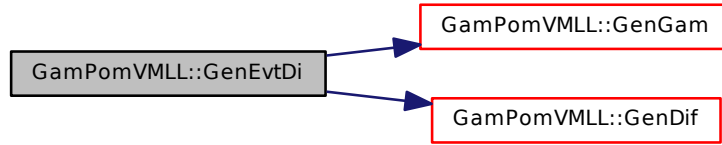
Here is the call graph for this function:



6.4.1.6 void GamPomVMLL::GenEvtDi () [private]

Generate a diffractive vector meson production event

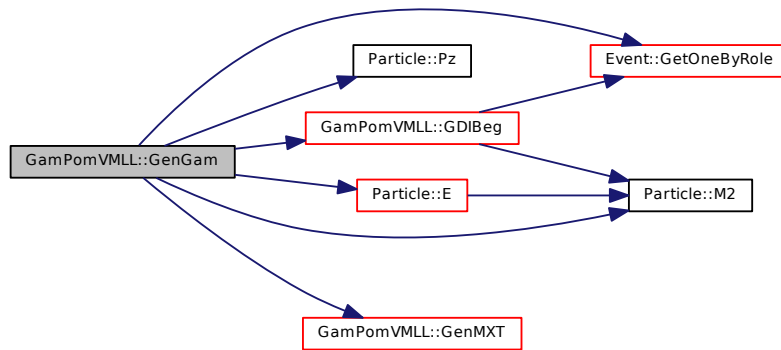
Here is the call graph for this function:



6.4.1.7 void GamPomVMLL::GenGam () [private]

Take 5-vectors of colliding e and p and generate a virtual photon, the momentum transfer of the pomeron, and the diffractive masses at the p and VM vertices

Here is the call graph for this function:



6.4.1.8 double GamPomVMLL::GenMXT (double * wght) [private]

Generate m_X^p , m_X^{VM} and t and determine if the generated combination is kinematically allowed

Author

Benno List

Date

18 Jul 1993 (MXTGEN)
10 May 1994 (GENMXT)

Here is the call graph for this function:



6.4.1.9 void GamPomVMLL::GenPhot (**Particle** * phot_, **Particle** * ele_, double * q2_, **Particle** pel_, double emin_, double q2max_) [private]

Generate photon with energy between *emin_* and electron energy and Q^2 less than *q2max_*, calculate 5-vector of scattered electron.

Parameters

in	<i>emin_</i>	Minimal allowed energy
in	<i>q2max_</i>	Maximal allowed Q^2
in	<i>pel_</i>	Incoming electron kinematics
out	<i>phot_</i>	Photon kinematics (mass is set to $-\sqrt{Q^2}$)
out	<i>ele_</i>	Outgoing electron kinematics
out	<i>q2_</i>	Absolute Photon momentum squared (positive!)

Author

Benno List

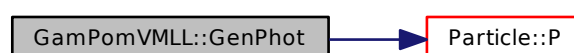
Date

22 Jan 1993

Note

Up to now only real photons with 1/k spectrum
Sets *_ihel*, *_ftrans*, *_epsil*

Here is the call graph for this function:



6.4.1.10 void GamPomVMLL::GEPhot (double * q2_, int * heli_) [private]

Generate one event with unweighted photon & electron

- according to WWA :

- transversal photonspectrum. $Q^2 \rightarrow 0$:

$$P(y, Q^2) = \frac{\alpha}{2\pi} \frac{1}{Q^2 y} \left(2(1-y) \left(1 - \frac{Q_{\min}^2}{Q^2} \right) + y^2 \right)$$

- longitudinal photonspectrum. $Q^2 \rightarrow 0$:

$$P(y, Q^2) = \frac{\alpha}{2\pi} \frac{1}{Q^2 y} (2(1-y))$$

- full transversal photonspectrum given by [1], [5], [6]
- full transversal and longitudinal spectrum by [1]
- calculate integrated factor over the spectrum: kinematical bounds : $[Y_{\min}, Y_{\max}] (W_{\min}), [Q_{\min}^2, Q_{\max}^2] (Q_{\text{cutoff}}^2)$

Parameters

out	q2_	Virtuality of photon (positive!): $Q^2 = -q^2$
out	heli_	Photon helicity: 0: longitudinal, 1, -1: transverse polarization

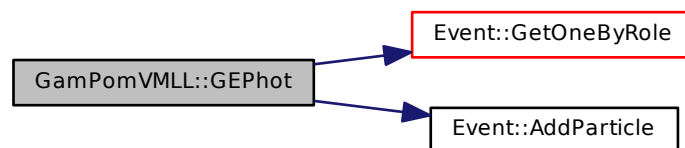
Author

T. Jansen

Date

6 Apr 1993

Here is the call graph for this function:



6.4.1.11 **Event*** Process::GetEvent () [inline], [inherited]

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

6.4.1.12 `bool Process::IsKinematicsDefined () [inline], [inherited]`

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:



6.4.1.13 `virtual bool Process::SetIncomingParticles (Particle ip1_, Particle ip2_) [inline], [virtual], [inherited]`

Specifies the incoming particles' kinematics as well as their properties using two [Particle](#) objects.

Parameters

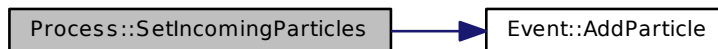
in	<i>ip1_</i>	Information on the first incoming particle
in	<i>ip2_</i>	Information on the second incoming particle

Returns

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

Reimplemented in [GamGamLL](#).

Here is the call graph for this function:



6.4.1.14 `virtual void Process::SetKinematics (Kinematics cuts_) [inline], [virtual], [inherited]`

Parameters

<code>in</code>	<code>cuts_</code>	The Cuts object containing the kinematic parameters
-----------------	--------------------	---

Reimplemented in [GamGamLL](#).

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

Parameters

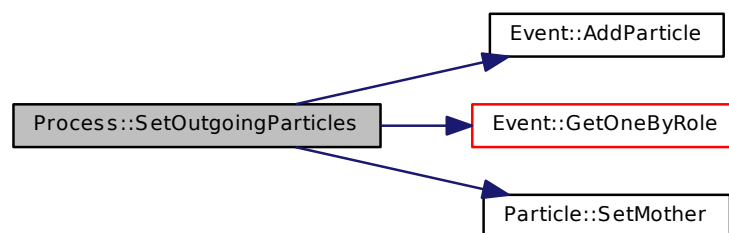
<code>in</code>	<code>part_</code>	Role of the particle in the process
<code>in</code>	<code>pdgId_</code>	Particle ID according to the PDG convention
<code>in</code>	<code>mothRole_</code>	Integer role of the outgoing particle's mother

Returns

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

Reimplemented in [GamGamLL](#).

Here is the call graph for this function:



6.4.1.16 `void Process::SetPoint (const unsigned int ndim_, double x_[]) [inherited]`

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

Parameters

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

6.4.1.17 `void GamPomVMLL::VMFlux () [private]`

Calculate relative photon luminosity for photon flux produced by *GEPhot*, weighted by VM propagator and cross section

Returns

`_genmxt_f` Total VM flux, relative to *e* flux
`_genmxt_df` Error of `_genmxt_f`
`_genmxt_ft` Relative VM flux for transverse VMs
`_genmxt_dft` Error of `_genmxt_ft`
`_genmxt_fl` Relative VM flux for longitudinal VMs
`_genmxt_dfl` Error of `_genmxt_fl`

Author

T. Jansen

Date

07 Apr 1993

6.4.1.18 double GamPomVMLL::VXMass (double mmin_, double mmax_) [private]

Generate hadronic mass between *mmin_* and *mmax_* for VM vertex

Parameters

in	<i>mmin_</i>	Minimal allowed mass
in	<i>mmax_</i>	Maximal allowed mass

Returns

Hadronic mass in GeV

Author

Benno List

Date

14 Jan 1992

6.4.2 Field Documentation

6.4.2.1 double GamPomVMLL::_alpha1 [private]

Note

Controls shrinkage of *b* slope

6.4.2.2 double GamPomVMLL::_amxb0 [private]

Note

For *_amxb0=0.0*, *_amxb0* is set according to production mode. Value is not meaningful for elastic VM production

6.4.2.3 double GamPomVMLL::_anexp [private]

Power law exponent.

- For *_anexp = 0* (default), a pure exponential spectrum is generated according to (taking $t < 0$)

$$\frac{d\sigma}{dt} = e^{bt}$$

- For *_anexp > 1*, an interpolated spectrum is generated according to

$$\frac{d\sigma}{dt} = \exp \left[-n \ln \left(-\frac{bt}{n} + 1 \right) \right] = \left(-\frac{bt}{n} + 1 \right)^{-n}$$

with $n = _anexp$

- Limit for small bt :

$$\exp(bt + ct^2)$$

with $c = b^2/2n$

- Limit for large $bt \gg n$: t^{-n}

6.4.2.4 double GamPomVMMLL::_b0 [private]

Slope parameter b of t distribution in GeV^{-2} at CM energy $_wb0$ and (for diffractive dissociation) mass $_amxb0$

Note

Must be positive!

6.4.2.5 double GamPomVMMLL::_br [private]

Branching ratio of the chosen decay channel. Useful values:

- 1 for $itypel = 0$
- 0.99 for $\rho^0 \rightarrow \pi^+\pi^-$
- 0.0221 for $\omega \rightarrow \pi^+\pi^-$
- 0.491 for $\phi \rightarrow K^+K^-$
- 0.344 for $\phi \rightarrow K_L^0 K_S^0$
- 0.0598 for $J/\psi \rightarrow e^+e^-, \mu^+\mu^-$
- 0.0425 for $\psi' \rightarrow \ell^+\ell^- X$ (including cascade decays)
- 0.025 for $\Upsilon(1s) \rightarrow \ell^+\ell^-$
- 0.02 for $\Upsilon(2s) \rightarrow \ell^+\ell^- X$ (including cascade decays)
- 0.0217 for $\Upsilon(3s) \rightarrow \ell^+\ell^- X$ (including cascade decays)

6.4.2.6 double GamPomVMMLL::_chi [private]

See $_xi$. χ is a purely phenomenological parameter with no theoretical justification!

6.4.2.7 double GamPomVMMLL::_cthelb [private]

Minimal $\cos\theta$ of scattered electron

6.4.2.8 double GamPomVMMLL::_eelmin [private]

Minimal energy of scattered electron in GeV

6.4.2.9 double GamPomVMMLL::_egamma [private]

Energy of photon in GeV for $_igammd = -1$

6.4.2.10 double GamPomVMMLL::_eprop [private]

Propagator term exponent ϵ_{prop} (see $_lambda$)

6.4.2.11 double GamPomVMMLL::_epsilm [private]

Note

Controls M_X spectrum

6.4.2.12 double GamPomVMMLL::_epsilw [private]

Note

Controls rise of $\sigma_{\gamma p}$ with W

6.4.2.13 int GamPomVMML::_igammd [private]

Photon generator mode. Possible values:

- -1: Fixed photon energy *_egamma*
- 0: $\frac{1}{k}$ spectrum
- 1: WWA/EPA approximation (including electron-mass effect and longitudinal flux). **Recommended**
- 2: Transverse spectrum *a la* [1]
- 3: Transverse & longitudinal spectrum *a la* [1]
- 4: as 3, but flux in proton rest frame

6.4.2.14 double GamPomVMML::_lambda [private]

Parameter for Q^2 -dependence of cross section in GeV:

$$\sigma(Q^2) = \frac{\sigma(0)}{\left(1 + \frac{Q^2}{\Lambda^2}\right)^{\epsilon_{\text{prop}}}}$$

6.4.2.15 double GamPomVMML::_wmax [private]

Note

If too low, it is set to \sqrt{s}

6.4.2.16 double GamPomVMML::_xi [private]

Parameter for Q^2 -dependence of σ_L/σ_T :

$$\frac{\sigma_L(Q^2)}{\sigma_T(Q^2)} = \frac{\xi Q^2/m^2}{1 + \xi \chi Q^2/m^2}$$

with

- $\frac{\sigma_L}{\sigma_T} \rightarrow \xi \frac{Q^2}{m^2}$ for low Q^2
- $\frac{\sigma_L}{\sigma_T} \rightarrow \frac{1}{\chi}$ for high Q^2

_xi is assumed to be less than 4 (more precisely, it is assumed that $\sigma_L(Q^2)$ is always less than $\sigma_T(0)$).

6.4.2.17 double GamPomVMML::_amassv [private]

Minimal mass of diffractive VM state. If value is too small (smaller than m_{π^0}), sets value to $m_{\text{VM}} + \text{some offset}$

Note

Value is only meaningful for *ifragv* = 1 or 2 or 955.

6.4.2.18 double GamPomVMML::_deminp [private]

Minimal energy released in decay of diffractive proton state, in GeV If value is too small (smaller than m_{π^0}), sets value to $0.236 = m_n + m_{\pi^0} + 0.10 - m_p$

Note

Value is only meaningful for *ifragp* = 1 or 2.

6.4.2.19 int GamPomVMML::ifragp [private]

Fragmentation mode for diffractive proton state. Possible values:

- 0 : Elastic scattering of proton
- 1 : Fragmentation by JETSET 7.3 with gluon emission (called DIFFVMg, contributed by Leszek Adamczyk)
- 1 : Fragmentation by JETSET 7.3
- 2 : Isotropic phase space decay into nucleon and pions
- 12212 : Elastic $N(1440)^+$ production at p vertex (for other N^* states, insert respective PDG code)

6.4.2.20 ParticleId GamPomVMML::ifragv [private]

Fragmentation mode for diffractive vector meson state. Possible values:

- 0 : Elastic vector meson production
- 1 : Fragmentation by JETSET 7.3
- 2 : Isotropic phase space decay into VM+pions
- 995 : diffractive pomeron-VM scattering (glueball production) see P.E.SCHLEIN (1994): Phys. Lett. B332, 136-140.

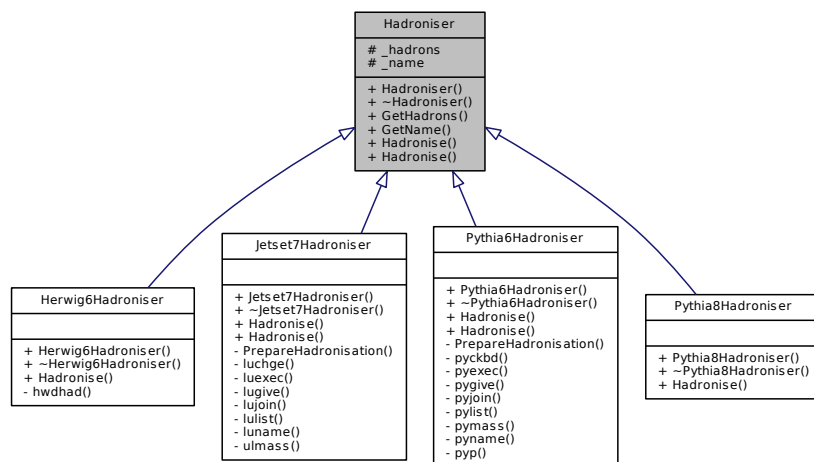
6.4.2.21 ParticleId GamPomVMML::itypvm [private]

Type of vector meson (should have $J^{PC} = 1^{--}$) to produce, and decay mode Possible values:

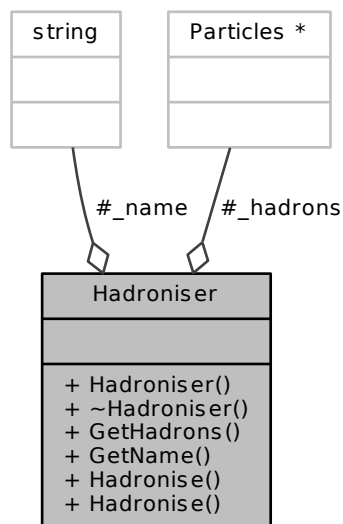
- 113 : ρ
- 223 : ω
- 333 : ϕ
- 443 : J/ψ
- 20443 : ψ'
- 553 : $\Upsilon(1s)$
- 20553 : $\Upsilon(2s)$
- 30553 : $\Upsilon(3s)$
- 40113 : $\rho(1450) \rightarrow \pi^+\pi^-\rho^0$
- 10333 : $\phi(1680) \rightarrow K\bar{K}$
- 22 : *diff. gamma dissoc. (special value)*

6.5 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.

6.5.1 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

6.5.2 Member Function Documentation

6.5.2.1 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

6.5.2.2 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 occurred successfully

Reimplemented in [Pythia6Hadroniser](#), and [Jetset7Hadroniser](#).

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

Launches the hadroniser on the full event information

Parameters

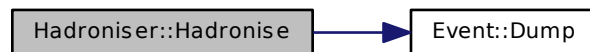
in,out	ev_	The event to hadronise
------------------------	---------------------	------------------------

Returns

A boolean stating whether or not the hadronisation occurred successfully

Reimplemented in [Pythia6Hadroniser](#), [Herwig6Hadroniser](#), [Jetset7Hadroniser](#), and [Pythia8Hadroniser](#).

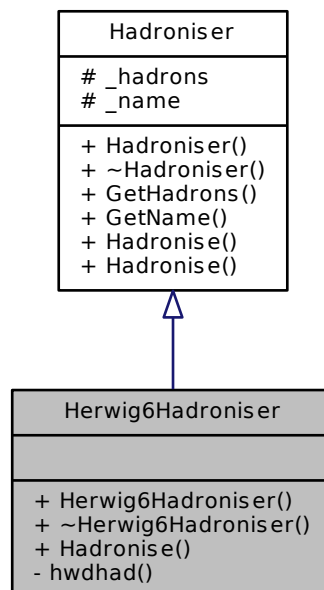
Here is the call graph for this function:



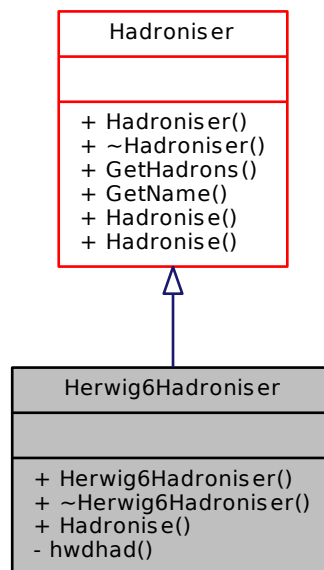
6.6 Herwig6Hadroniser Class Reference

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

6.6.1 Member Function Documentation

6.6.1.1 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

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

Parameters

in	<i>part_</i>	The Particle object which will be hadronised
-----------	--------------	--

Returns

A boolean stating whether or not the hadronisation occurred successfully

Reimplemented in [Pythia6Hadroniser](#), and [Jetset7Hadroniser](#).

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

Launches the hadroniser on the full event information

Parameters

in,out	<i>ev_</i>	The event to hadronise
---------------	------------	------------------------

Returns

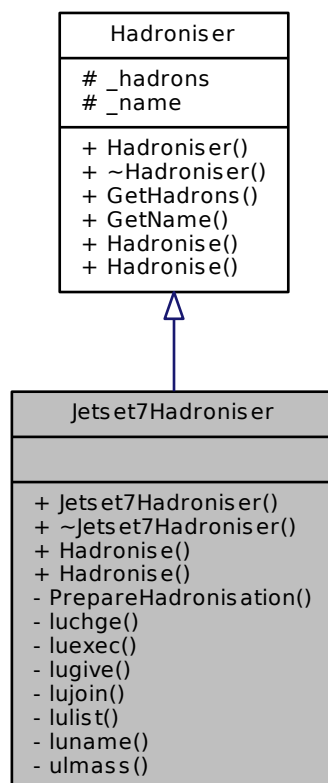
A boolean stating whether or not the hadronisation occurred successfully

Reimplemented from [Hadroniser](#).

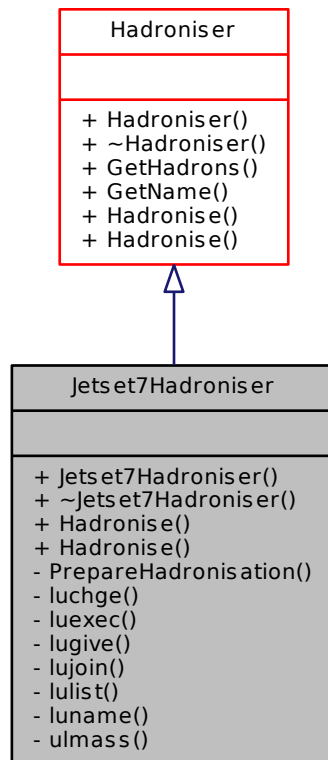
6.7 Jetset7Hadroniser Class Reference

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_)

6.7.1 Member Function Documentation

6.7.1.1 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

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

Parameters

in	<i>part_</i>	The Particle object which will be hadronised
----	--------------	--

Returns

A boolean stating whether or not the hadronisation occurred successfully

Reimplemented from [Hadroniser](#).

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

Launches the hadroniser on the full event information

Parameters

in,out	<i>ev_</i>	The event to hadronise
--------	------------	------------------------

Returns

A boolean stating whether or not the hadronisation occurred successfully

Reimplemented from [Hadroniser](#).

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

Give the charge for a parton/particle

Parameters

in	<i>pdgid_</i>	PDG id of the parton/particle
----	---------------	-------------------------------

6.7.1.5 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.

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

Set the value of any variable residing in the commonblocks 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
----	-------	--

6.7.1.7 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 <i>njoin_</i> . The <i>njoin_</i> 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.

6.7.1.8 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 parentheses. 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 MD-CY).

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	<i>mlist_</i>	Determines what is to be listed (see detailed description)
----	---------------	--

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

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

Parameters

in	<i>pdgid_</i>	PDG id of the parton/particle
----	---------------	-------------------------------

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

Gives the mass for a parton/particle

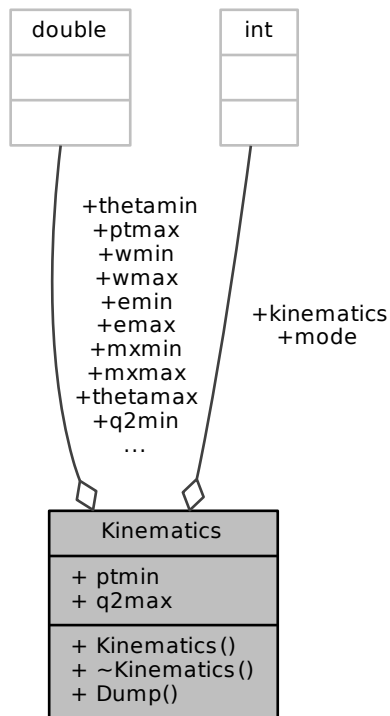
Parameters

in	<i>pdgid_</i>	PDG id of the parton/particle
----	---------------	-------------------------------

6.8 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 `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 (θ_{\max}) angle of the outgoing leptons, expressed in degrees.

- double `thetamin`

Minimal polar (θ_{\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.

6.8.1 Field Documentation

6.8.1.1 int Kinematics::kinematics

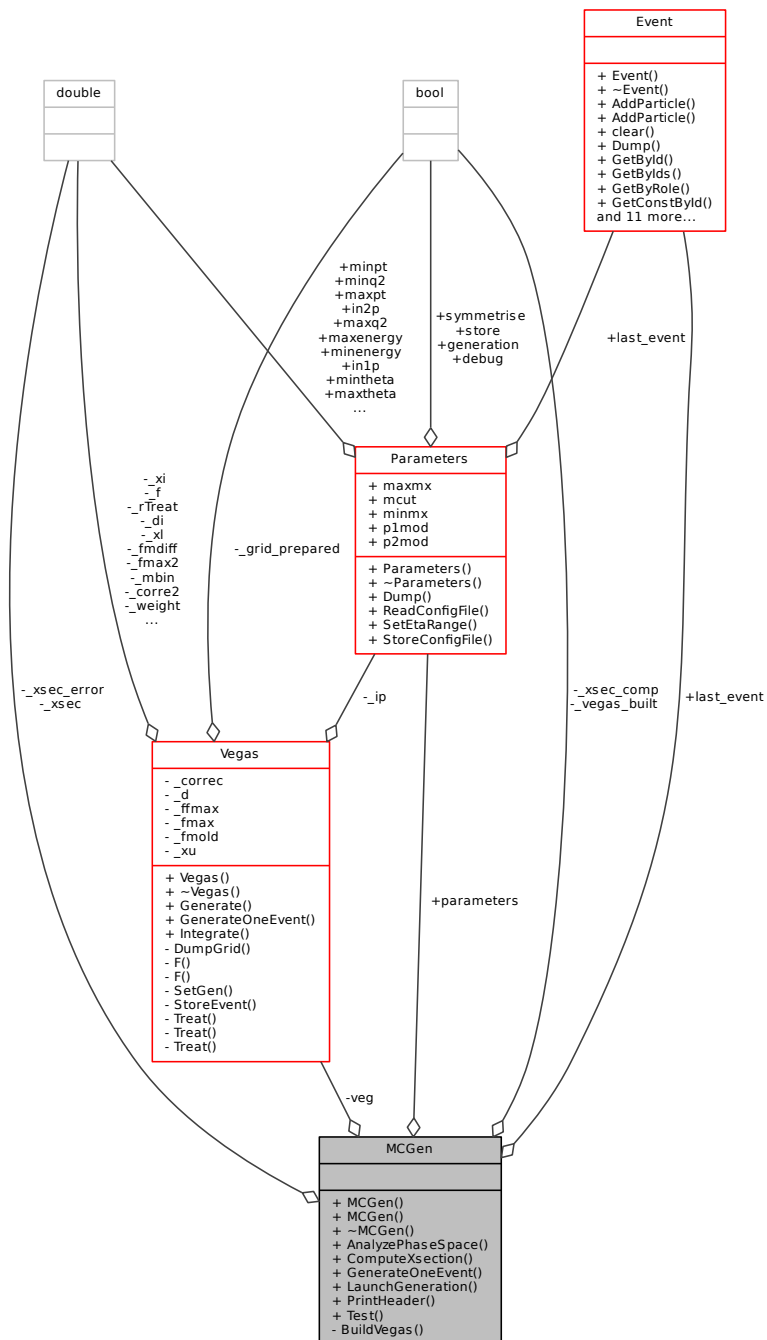
Type of kinematics to consider for the process. Can either be :

- 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

6.9 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.

- void `Test` ()

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 been 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.

6.9.1 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

6.9.2 Constructor & Destructor Documentation

6.9.2.1 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

6.9.2.2 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	<i>ip_</i>	List of input parameters defining the phase space on which to perform the integration
-----------	------------	---

6.9.3 Member Function Documentation

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

Returns

The Parameter object embedded in this class

6.9.3.2 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	<i>xsec_</i>	The computed cross-section, in pb
out	<i>err_</i>	The absolute integration error on the computed cross-section, in pb

6.9.3.3 **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

6.9.3.4 void MCGen::LaunchGeneration ()

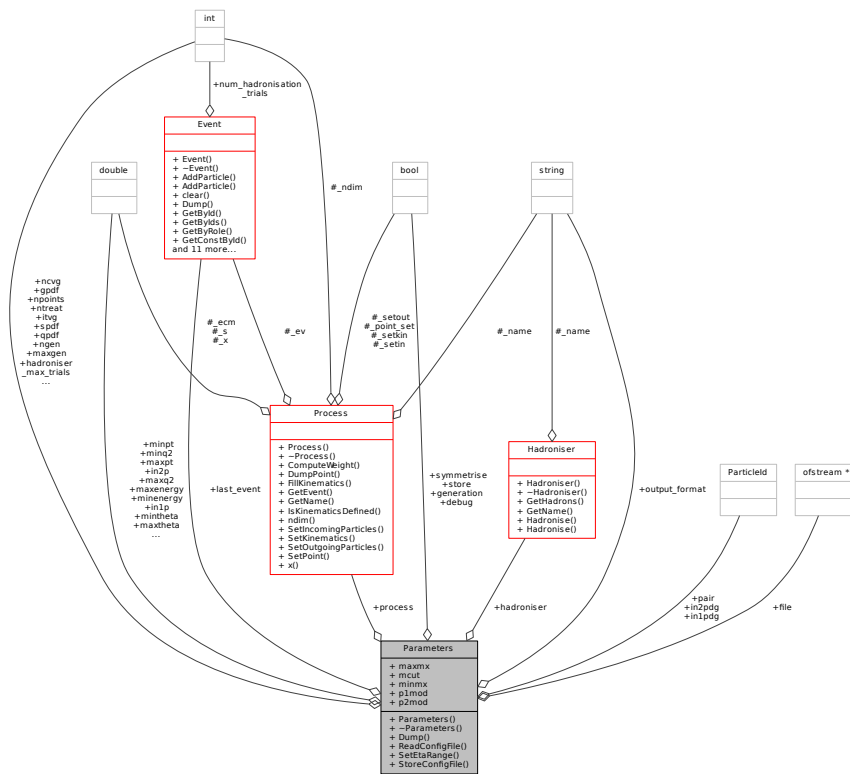
Launches the full events generation

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

6.10 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 `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 `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.
- int `p1mod`
First particle's mode.
- int `p2mod`
Second particle's mode.
- 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 qpdf`
Number of quarks.
- `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 ?

6.10.1 Detailed Description

Note

The default parameters are derived from GMUINI in LPAIR

6.10.2 Member Function Documentation

6.10.2.1 `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

<code>in</code>	<code>inFile_</code>	Name of the configuration file to load
-----------------	----------------------	--

Returns

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

6.10.2.2 `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

<code>in</code>	<code>etamin_</code>	The minimal value of η for the outgoing leptons
<code>in</code>	<code>etamax_</code>	The maximal value of η for the outgoing leptons

6.10.2.3 `bool Parameters::StoreConfigFile (std::string outFile_)`

Parameters

<code>in</code>	<code>outFile_</code>	Name of the configuration file to create
-----------------	-----------------------	--

Returns

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

6.10.3 Field Documentation

6.10.3.1 `bool Parameters::debug`

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

6.10.3.2 double Parameters::maxmx

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

6.10.3.3 double Parameters::maxpt

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

6.10.3.4 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 $p_T \geq 1 \text{ GeV}/c$, or
 - $0.75 < \frac{|p_z|}{|\mathbf{p}|} \leq 0.95$ and $p_z > 1 \text{ GeV}/c$,
- 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

6.10.3.5 double Parameters::minmx

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

6.10.3.6 double Parameters::minpt

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

6.10.3.7 int Parameters::ntreat

Note

Is it correctly implemented ?

6.10.3.8 int Parameters::p1mod

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 in ILPAIR

6.10.3.9 int Parameters::p2mod

Note

Was named EMOD in ILPAIR

6.10.3.10 ParticleId Parameters::pair

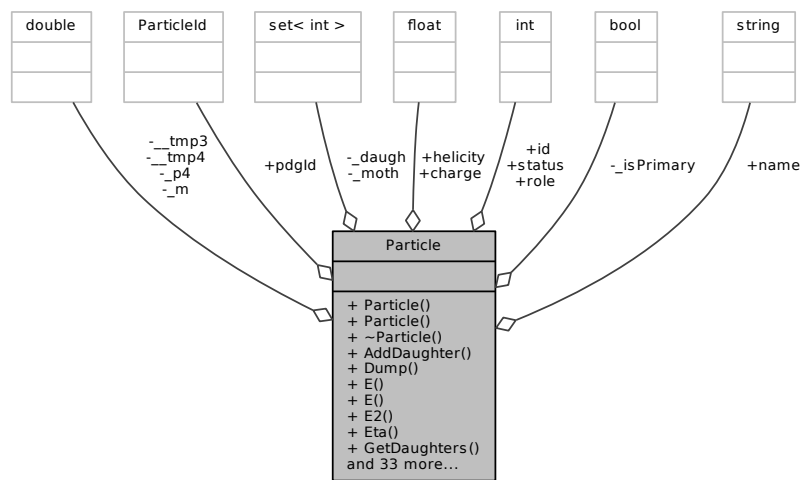
The particle code of produced leptons, as defined by the PDG convention :

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

6.11 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 GeV².
- 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)`
- `ParticlesIds 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 &)`
Subtracts 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]
- ParticleIds [_daugh](#)
List of daughter particles.
- bool [_isPrimary](#)
Is the particle a primary particle ?
- double [_m](#)
Mass in GeV/c^2 .
- ParticleIds [_moth](#)
List of mother particles.
- double [_p4](#) [4]

6.11.1 Detailed Description

Kinematic information for one particle

6.11.2 Member Function Documentation

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

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

Parameters

<code>in</code>	<code>part_</code>	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

6.11.2.2 `void Particle::Dump ()`

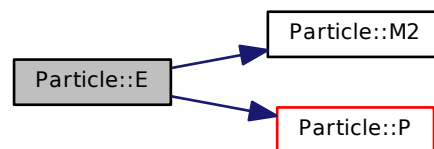
Dumps into the standard output stream all the available information on this particle

6.11.2.3 `void Particle::E (double E_) [inline]`

Parameters

<code>in</code>	<code>E_</code>	Energy, in GeV
-----------------	-----------------	----------------

Here is the call graph for this function:



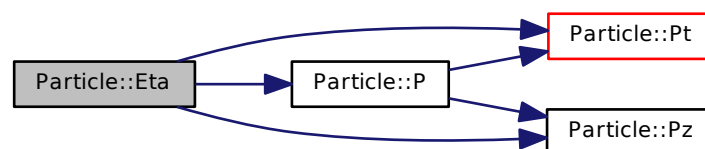
6.11.2.4 `double Particle::Eta () [inline]`

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:



6.11.2.5 `std::vector<int> Particle::GetDaughters ()`

Returns

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

6.11.2.6 `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	<i>revert_</i>	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

6.11.2.7 `ParticleIds Particle::GetMothersIds () const [inline]`

Returns

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

6.11.2.8 `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	<i>algo_</i>	Algorithm in use to hadronise the particle
-----------	--------------	--

Returns

A boolean stating whether or not the particle has been hadronised

6.11.2.9 `double* Particle::LorentzBoost (double p_[3])`

Lorentz boost from ROOT

6.11.2.10 `double Particle::M () const [inline]`

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

Returns

The particle's mass

6.11.2.11 `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

<i>m_</i>	The mass in GeV/c^2 to set
-----------	-------------------------------------

Returns

A boolean stating whether or not the mass was correctly set

6.11.2.12 `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

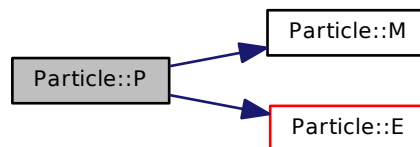
Parameters

<code>in</code>	<code>px_</code>	Momentum along the x -axis, in GeV/c
<code>in</code>	<code>py_</code>	Momentum along the y -axis, in GeV/c
<code>in</code>	<code>pz_</code>	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:



6.11.2.13 `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.

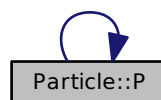
Parameters

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

Returns

A boolean stating the validity of the particle's kinematics

Here is the call graph for this function:



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

Parameters

<code>in</code>	<code>p_</code>	4-momentum
-----------------	-----------------	------------

Returns

A boolean stating the validity of the particle's kinematics

Here is the call graph for this function:



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

Get one component of the particle's 4-momentum

Parameters

<code>in</code>	<code>c_</code>	The component to retrieve: <ul style="list-style-type: none"> ▪ 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:

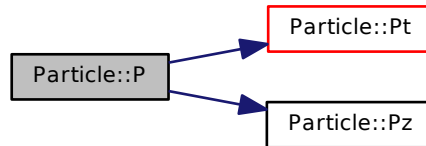


6.11.2.16 `double Particle::P () const [inline]`

Returns

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

Here is the call graph for this function:



6.11.2.17 `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:



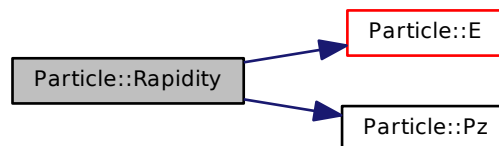
6.11.2.18 `double Particle::Rapidity () [inline]`

Computes and returns y , the rapidity of the particle

Returns

The rapidity of the particle

Here is the call graph for this function:



6.11.2.19 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
----	-------	---

6.11.3 Field Documentation

6.11.3.1 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)

6.11.3.2 float Particle::helicity

[Particle](#)'s helicity Float??

6.11.3.3 ParticleId Particle::pdgId

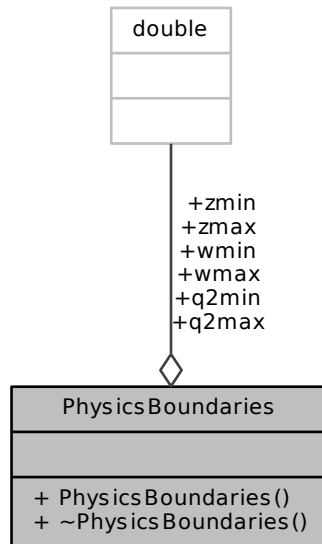
Unique identifier for a particle type. From [2] : *The Monte Carlo particle numbering scheme [...] is intended to facilitate interfacing between event generators, detector simulators, and analysis packages used in particle physics.*

6.11.3.4 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

6.12 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 .

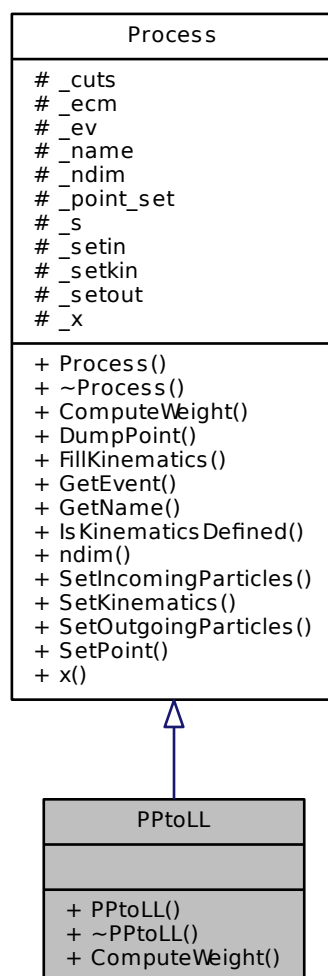
6.12.1 Detailed Description

List of physical constraints to apply on the phase space

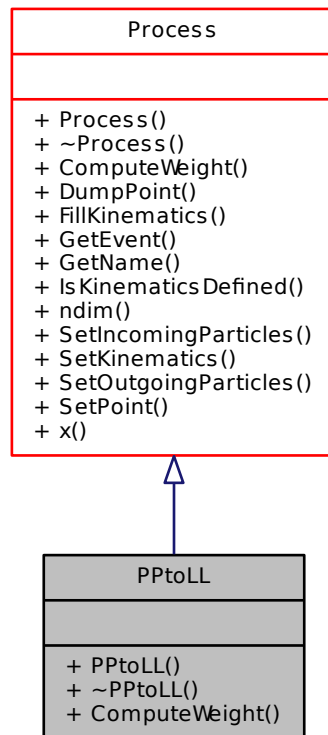
6.13 PPtoLL Class Reference

Computes the matrix element for a CE $\gamma\gamma \rightarrow \ell^+\ell^-$ process using k_T -factorization approach.

Inheritance diagram for PPtoLL:



Collaboration diagram for PPtoLL:



Public Member Functions

- 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.
- 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 GeV²
- 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.

6.13.1 Member Function Documentation

6.13.1.1 virtual void [Process::FillKinematics](#) (bool symmetrise_ = false) [inline], [virtual], [inherited]

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) ?
--------------------	-----------------------------	--

Reimplemented in [GamGamLL](#), and [GamPomVMMLL](#).

6.13.1.2 [Event*](#) [Process::GetEvent](#) () [inline], [inherited]

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

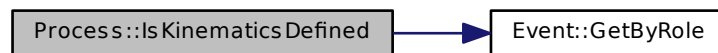
6.13.1.3 `bool Process::IsKinematicsDefined () [inline], [inherited]`

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:



6.13.1.4 `virtual bool Process::SetIncomingParticles (Particle ip1_, Particle ip2_) [inline], [virtual], [inherited]`

Specifies the incoming particles' kinematics as well as their properties using two [Particle](#) objects.

Parameters

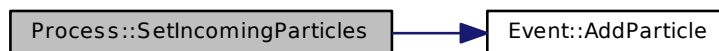
<code>in</code>	<code>ip1_</code>	Information on the first incoming particle
<code>in</code>	<code>ip2_</code>	Information on the second incoming particle

Returns

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

Reimplemented in [GamGamLL](#).

Here is the call graph for this function:



6.13.1.5 `virtual void Process::SetKinematics (Kinematics cuts_) [inline], [virtual], [inherited]`

Parameters

in	<i>cuts_</i>	The Cuts object containing the kinematic parameters
-----------	--------------	---

Reimplemented in [GamGamLL](#).

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

Parameters

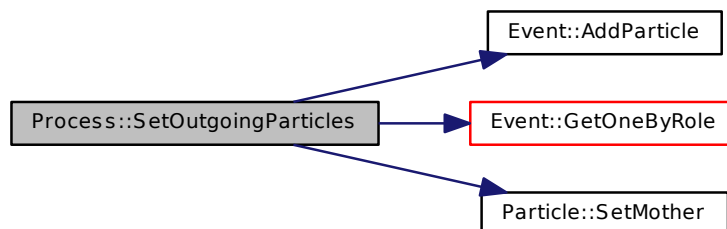
in	<i>part_</i>	Role of the particle in the process
in	<i>pdgId_</i>	Particle ID according to the PDG convention
in	<i>mothRole_</i>	Integer role of the outgoing particle's mother

Returns

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

Reimplemented in [GamGamLL](#).

Here is the call graph for this function:



6.13.1.7 `void Process::SetPoint (const unsigned int ndim_, double x_[]) [inherited]`

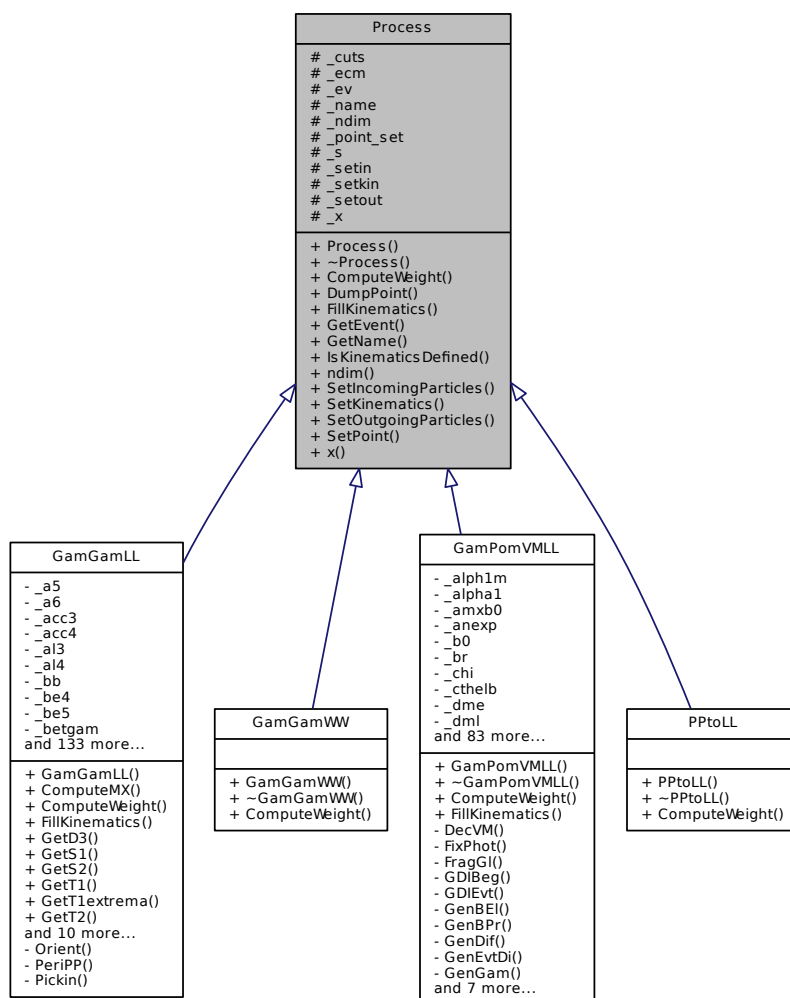
Sets the phase space point to compute the weight associated to it.

Parameters

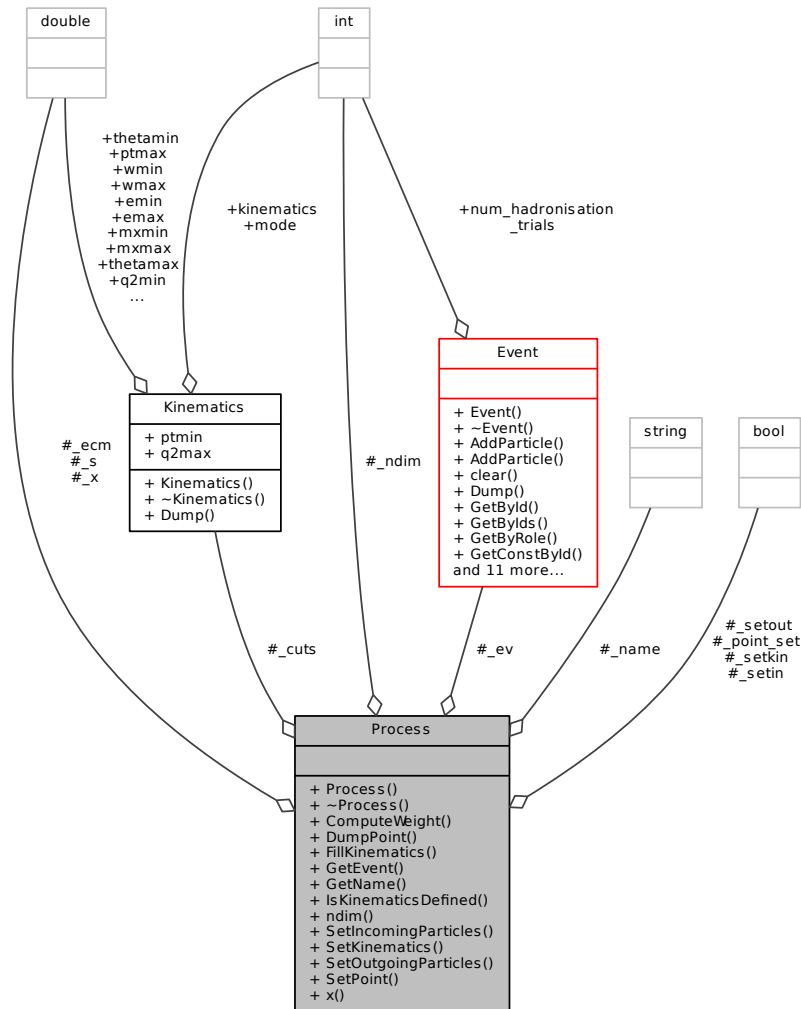
in	<i>ndim_</i>	The number of dimensions of the point in the phase space
in	<i>x_[]</i>	The (<i>ndim_</i>)-dimensional point in the phase space on which the kinematics and the cross-section are computed

6.14 Process Class Reference

Inheritance diagram for Process:



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.
- 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 GeV²
- 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.

6.14.1 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

6.14.2 Member Function Documentation

6.14.2.1 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	<i>symmetrise_</i>	Do we have to symmetrise the event (randomise the production of the positively- and negatively-charged lepton) ?
-----------	--------------------	--

Reimplemented in [GamGamLL](#), and [GamPomVMLL](#).

6.14.2.2 **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

6.14.2.3 **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:



6.14.2.4 **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

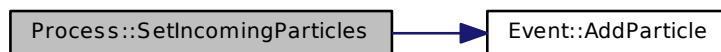
in	<i>ip1_</i>	Information on the first incoming particle
in	<i>ip2_</i>	Information on the second incoming particle

Returns

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

Reimplemented in [GamGamLL](#).

Here is the call graph for this function:



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

Parameters

in	<i>cuts_</i>	The Cuts object containing the kinematic parameters
----	--------------	---

Reimplemented in [GamGamLL](#).

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

Parameters

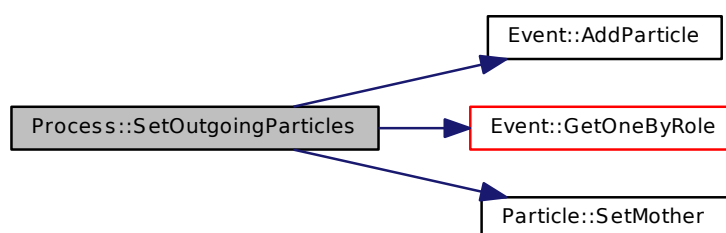
in	<i>part_</i>	Role of the particle in the process
in	<i>pdgId_</i>	Particle ID according to the PDG convention
in	<i>mothRole_</i>	Integer role of the outgoing particle's mother

Returns

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

Reimplemented in [GamGamLL](#).

Here is the call graph for this function:



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

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

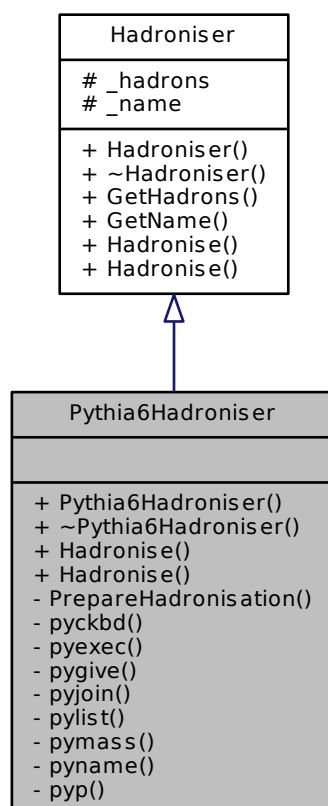
Parameters

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

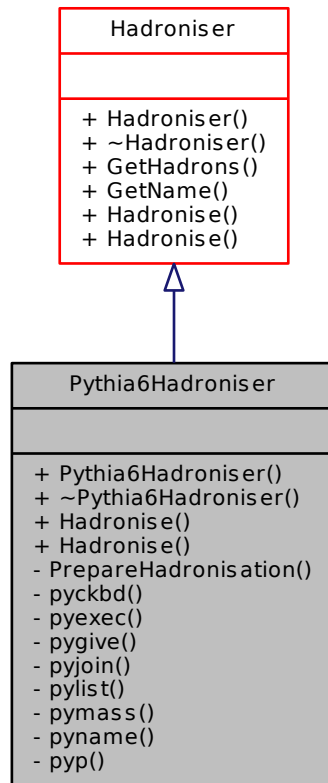
6.15 Pythia6Hadroniser Class Reference

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_)

6.15.1 Detailed Description

Full interface to the Pythia6 [4] 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.

6.15.2 Member Function Documentation

6.15.2.1 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

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

Parameters

in	<i>part_</i>	The Particle object which will be hadronised
-----------	--------------	--

Returns

A boolean stating whether or not the hadronisation occurred successfully

Reimplemented from [Hadroniser](#).

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

Launches the hadroniser on the full event information

Parameters

in,out	<i>ev_</i>	The event to hadronise
---------------	------------	------------------------

Returns

A boolean stating whether or not the hadronisation occurred successfully

Reimplemented from [Hadroniser](#).

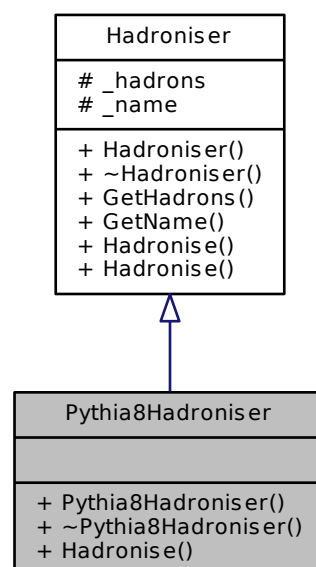
6.15.2.4 static void Pythia6Hadroniser::pyjoin (int njoin_, int ijoin_[2]) [inline], [static], [private]

Parameters

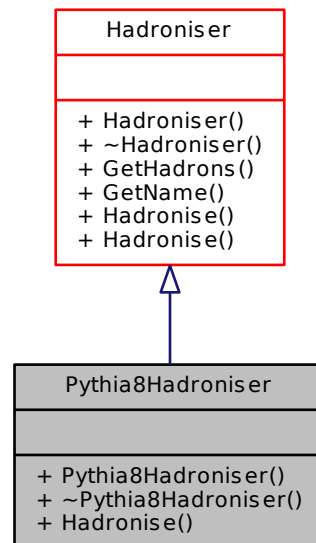
in	<i>njoin_</i>	Number of particles to join in the colour flow
in	<i>ijoin_</i>	List of particles unique identifier to join in the colour flow

6.16 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.

6.16.1 Member Function Documentation

6.16.1.1 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

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

Launches the hadroniser on the full event information

Parameters

<code>in,out</code>	<code>ev_</code>	The event to hadronise
---------------------	------------------	------------------------

Returns

A boolean stating whether or not the hadronisation occurred successfully

Reimplemented from [Hadroniser](#).

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

Parameters

<code>in</code>	<code>part_</code>	The Particle object which will be hadronised
-----------------	--------------------	--

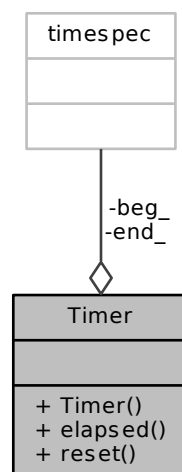
Returns

A boolean stating whether or not the hadronisation occurred successfully

Reimplemented in [Pythia6Hadroniser](#), and [Jetset7Hadroniser](#).

6.17 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.

6.17.1 Detailed Description

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

6.17.2 Member Function Documentation

6.17.2.1 `double Timer::elapsed () [inline]`

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

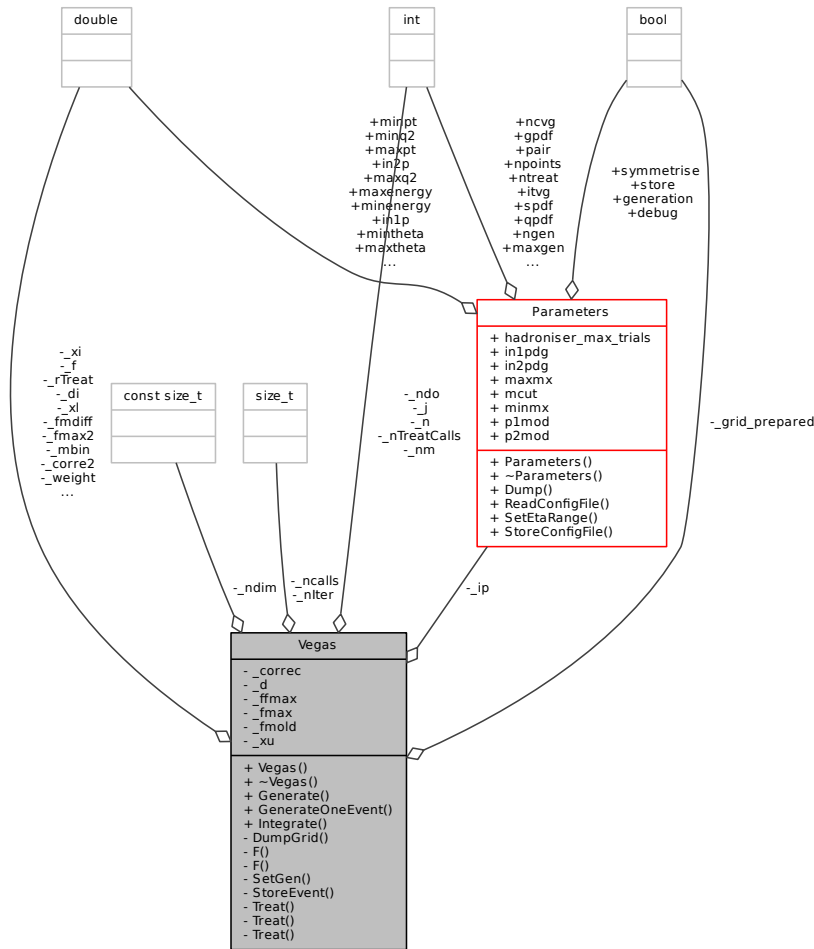
Returns

The elapsed time in seconds

6.18 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_)

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 [__corre2](#)
- double [__correc](#)
- double * [__d](#) [MAX_ND]
- double * [__di](#) [MAX_ND]
- 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 [__grid_prepared](#)
Flag to define whether or not the grid has been prepared using SetGen (very time-consuming operation, thus needs to be called once)
- [Parameters](#) * [__ip](#)
List of parameters to specify the integration range and the physics determining the phase space.
- int [__j](#)
Selected bin at which the function will be evaluated.
- double [__mbin](#)
Integration grid size parameter.
- int * [__n](#)
- size_t [__ncalls](#)
Fixed number of function calls to use.
- const size_t [__ndim](#)
The number of dimensions on which to integrate the function.
- unsigned int [__ndo](#)
- size_t [__nlter](#)
Number of points to generate in order to integrate the function.
- int * [__nm](#)
- int [__nTreatCalls](#)
- double [__rTreat](#)
- double [__weight](#)
Weight of the point in the total integration.
- double * [__xi](#) [MAX_ND]
- double * [__xl](#)
Lower bounds for the points to generate.
- double * [__xu](#)
Upper bounds for the points to generate.

6.18.1 Detailed Description

Main occurrence of the Monte-Carlo integrator[3] developed by G.P. Lepage in 1978

6.18.2 Constructor & Destructor Documentation

6.18.2.1 `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 [3]

Parameters

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

6.18.3 Member Function Documentation

6.18.3.1 `void Vegas::DumpGrid () [private]`

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

6.18.3.2 `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	<i>x_</i>	The point at which the function is to be evaluated
-----------	-----------	--

Returns

Function value at this point *x_*

6.18.3.3 `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	<i>x_</i>	The point at which the function is to be evaluated
in	<i>ip_</i>	A set of parameters to fully define the function

Returns

Function value at this point *x_*

6.18.3.4 `void Vegas::Generate ()`

Launches the [Vegas](#) generation of events according to the provided input parameters.

6.18.3.5 `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)

6.18.3.6 `int Vegas::Integrate (double * result_, double * abserr_)`

[Vegas](#) algorithm to perform the (`__dim`)-dimensional Monte Carlo integration of a given function as described in [3]

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	<i>result_</i>	The cross section as integrated by Vegas for the given phase space restrictions
out	<i>abserr_</i>	The error associated to the computed cross section

Returns

0 if the integration was performed successfully

6.18.3.7 `void Vegas::SetGen () [private]`

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

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

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

Parameters

in	<i>x_</i>	The <i>__ndim</i> -dimensional point in the phase space defining the unique event to store
----	-----------	--

Returns

A boolean stating whether or not the event could be saved

6.18.3.9 `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	<i>x_</i>	The <i>__ndim</i> -dimensional point at which the stabilised function is to be evaluated
in	<i>ip_</i>	The physics parameters to apply to the function to evaluate
in	<i>storedbg_</i>	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_*

6.18.3.10 `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:



6.18.3.11 `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:



6.18.4 Field Documentation

6.18.4.1 `double(* Vegas::_f)(double * $x_{_}$, size_t ndim_, void *params_) [private]`

Parameters

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

6.18.4.2 `int Vegas::_nTreatCalls` [private]

Has the `Treat` function already been called once ?

6.18.4.3 `double Vegas::_rTreat` [private]

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

References

- [1] I. Abt and John Rodgers Smith. MC upgrades to study untagged events. h1-10/92-249, 1992. [40](#), [45](#)
- [2] J. Beringer et al. Review of Particle Physics (RPP). *Phys.Rev.*, D86:010001, 2012. [75](#)
- [3] G. Peter Lepage. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, 27(2):192 – 203, 1978. [1](#), [98](#), [99](#)
- [4] Torbjorn Sjostrand, Stephen Mrenna, and Peter Z. Skands. PYTHIA 6.4 Physics and Manual. *JHEP*, 0605:026, 2006. [90](#)
- [5] John Rodgers Smith. An experimentalist's guide to photon flux calculations. h1-12/92-259, 1992. [40](#)
- [6] John Rodgers Smith. Photon fluxes with beam mass effects and polarizations. h1-01/94-338, 1994. [40](#)
- [7] J.A.M. Vermaseren. Two-photon processes at very high energies. *Nuclear Physics B*, 229(2):347 – 371, 1983. [1](#), [14](#), [17](#)

Index

- [_alpha1](#)
GamPomVMLL, [43](#)
 - [_amxb0](#)
GamPomVMLL, [43](#)
 - [_anexp](#)
GamPomVMLL, [43](#)
 - [_b0](#)
GamPomVMLL, [43](#)
 - [_br](#)
GamPomVMLL, [44](#)
 - [_chi](#)
GamPomVMLL, [44](#)
 - [_cthelb](#)
GamPomVMLL, [44](#)
 - [_delta](#)
GamGamLL, [24](#)
 - [_eelmin](#)
GamPomVMLL, [44](#)
 - [_egamma](#)
GamPomVMLL, [44](#)
 - [_eprop](#)
GamPomVMLL, [44](#)
 - [_epsilm](#)
GamPomVMLL, [44](#)
 - [_epsilw](#)
GamPomVMLL, [44](#)
 - [_f](#)
Vegas, [101](#)
 - [_igammd](#)
GamPomVMLL, [44](#)
 - [_lambda](#)
GamPomVMLL, [45](#)
 - [_nTreatCalls](#)
Vegas, [102](#)
 - [_p4](#)
Particle, [75](#)
 - [_part](#)
Event, [9](#)
 - [_rTreat](#)
Vegas, [102](#)
 - [_wmax](#)
GamPomVMLL, [45](#)
 - [_xi](#)
GamPomVMLL, [45](#)
- AddDaughter
Particle, [69](#)
- AddParticle
Event, [5](#)
- amassv
GamPomVMLL, [45](#)
- AnalyzePhaseSpace
MCGen, [62](#)
- ComputeMX
GamGamLL, [18](#)

- ComputeWeight
GamGamLL, [19](#)
- ComputeXsection
MCGen, [62](#)
- debug
Parameters, [65](#)
- DecVM
GamPomVMLL, [35](#)
- deminp
GamPomVMLL, [45](#)
- Dump
Event, [6](#)
Particle, [70](#)
- DumpGrid
Vegas, [98](#)
- E
Particle, [70](#)
- elapsed
Timer, [95](#)
- Eta
Particle, [70](#)
- Event, [3](#)
 - [_part](#), [9](#)
 - AddParticle, [5](#)
 - Dump, [6](#)
 - GetById, [6](#)
 - GetByIds, [6](#)
 - GetByRole, [6](#)
 - GetDaughters, [6](#)
 - GetMothers, [7](#)
 - GetOneByRole, [7](#)
 - GetParticles, [8](#)
 - GetRoles, [8](#)
 - GetStableParticles, [8](#)
 - np, [9](#)
 - NumParticles, [8](#)
 - Store, [8](#)
 - time_generation, [9](#)
 - time_total, [9](#)
- F
Vegas, [98](#)
- FillKinematics
GamGamLL, [19](#)
GamGamWW, [27](#)
GamPomVMLL, [35](#)
PPToLL, [79](#)
Process, [84](#)
- FixPhot
GamPomVMLL, [35](#)
- GDIBeg
GamPomVMLL, [36](#)
- GEPhot

- GamPomVMML, 39
- GamGamLL, 9
 - _delta, 24
 - ComputeMX, 18
 - ComputeWeight, 19
 - FillKinematics, 19
 - GamGamLL, 18
 - GamGamLL, 18
 - GetEvent, 20
 - GetT1, 20
 - GetT1extrema, 20
 - GetT2, 20
 - GetT2extrema, 20
 - IsKinematicsDefined, 21
 - Orient, 21
 - PeriPP, 21
 - Pickin, 22
 - PrepareHadronisation, 22
 - SetIncomingParticles, 22
 - SetKinematics, 23
 - SetOutgoingParticles, 23
 - SetPoint, 24
- GamGamWW, 24
 - FillKinematics, 27
 - GetEvent, 27
 - IsKinematicsDefined, 27
 - SetIncomingParticles, 28
 - SetKinematics, 28
 - SetOutgoingParticles, 28
 - SetPoint, 29
- GamPomVMML, 29
 - _alpha1, 43
 - _amxb0, 43
 - _anexp, 43
 - _b0, 43
 - _br, 44
 - _chi, 44
 - _cthelb, 44
 - _eelmin, 44
 - _egamma, 44
 - _eprop, 44
 - _epsilm, 44
 - _epsilw, 44
 - _igammd, 44
 - _lambda, 45
 - _wmax, 45
 - _xi, 45
 - amassv, 45
 - DecVM, 35
 - deminp, 45
 - FillKinematics, 35
 - FixPhot, 35
 - GDIBeg, 36
 - GEPhot, 39
 - GenDif, 37
 - GenEvtDi, 37
 - GenGam, 38
 - GenMXT, 38
 - GenPhot, 39
 - GetEvent, 40
 - ifragp, 45
 - ifragv, 46
 - IsKinematicsDefined, 40
 - itypvm, 46
 - SetIncomingParticles, 41
 - SetKinematics, 41
 - SetOutgoingParticles, 42
 - SetPoint, 42
 - VMFlux, 42
 - VXMass, 43
 - GenDif
 - GamPomVMML, 37
 - GenEvtDi
 - GamPomVMML, 37
 - GenGam
 - GamPomVMML, 38
 - GenMXT
 - GamPomVMML, 38
 - GenPhot
 - GamPomVMML, 39
 - Generate
 - Vegas, 98
 - GenerateOneEvent
 - MCGen, 62
 - Vegas, 98
 - GetById
 - Event, 6
 - GetByIds
 - Event, 6
 - GetByRole
 - Event, 6
 - GetDaughters
 - Event, 6
 - Particle, 70
 - GetEvent
 - GamGamLL, 20
 - GamGamWW, 27
 - GamPomVMML, 40
 - PPToLL, 79
 - Process, 85
 - GetHadrons
 - Hadroniser, 48
 - Herwig6Hadroniser, 50
 - Jetset7Hadroniser, 54
 - Pythia6Hadroniser, 90
 - Pythia8Hadroniser, 92
 - GetLHEline
 - Particle, 71
 - GetMothers
 - Event, 7
 - GetMothersIds
 - Particle, 71
 - GetOneByRole
 - Event, 7
 - GetParticles
 - Event, 8

- GetRoles
 - Event, [8](#)
- GetStableParticles
 - Event, [8](#)
- GetT1
 - GamGamLL, [20](#)
- GetT1extrema
 - GamGamLL, [20](#)
- GetT2
 - GamGamLL, [20](#)
- GetT2extrema
 - GamGamLL, [20](#)
- Hadronise
 - Hadroniser, [48](#)
 - Herwig6Hadroniser, [51](#)
 - Jetset7Hadroniser, [54](#)
 - Particle, [71](#)
 - Pythia6Hadroniser, [90](#)
 - Pythia8Hadroniser, [92](#), [94](#)
- Hadroniser, [47](#)
 - GetHadrons, [48](#)
 - Hadronise, [48](#)
- helicity
 - Particle, [75](#)
- Herwig6Hadroniser, [49](#)
 - GetHadrons, [50](#)
 - Hadronise, [51](#)
- ifragp
 - GamPomVMMLL, [45](#)
- ifragv
 - GamPomVMMLL, [46](#)
- Integrate
 - Vegas, [98](#)
- IsKinematicsDefined
 - GamGamLL, [21](#)
 - GamGamWW, [27](#)
 - GamPomVMMLL, [40](#)
 - PPtoLL, [79](#)
 - Process, [85](#)
- itypvm
 - GamPomVMMLL, [46](#)
- Jetset7Hadroniser, [51](#)
 - GetHadrons, [54](#)
 - Hadronise, [54](#)
 - luchge, [54](#)
 - luexec, [54](#)
 - lugive, [54](#)
 - lujoin, [55](#)
 - lulist, [55](#)
 - luname, [57](#)
 - ulmass, [57](#)
- Kinematics, [57](#)
 - kinematics, [59](#)
- kinematics
 - Kinematics, [59](#)
- LaunchGeneration
 - MCGen, [62](#)
- LorentzBoost
 - Particle, [71](#)
- luchge
 - Jetset7Hadroniser, [54](#)
- luexec
 - Jetset7Hadroniser, [54](#)
- lugive
 - Jetset7Hadroniser, [54](#)
- lujoin
 - Jetset7Hadroniser, [55](#)
- lulist
 - Jetset7Hadroniser, [55](#)
- luname
 - Jetset7Hadroniser, [57](#)
- M
 - Particle, [71](#)
- MCGen, [59](#)
 - AnalyzePhaseSpace, [62](#)
 - ComputeXsection, [62](#)
 - GenerateOneEvent, [62](#)
 - LaunchGeneration, [62](#)
 - MCGen, [62](#)
 - MCGen, [62](#)
- maxmx
 - Parameters, [65](#)
- maxpt
 - Parameters, [66](#)
- mcut
 - Parameters, [66](#)
- minmx
 - Parameters, [66](#)
- minpt
 - Parameters, [66](#)
- np
 - Event, [9](#)
- ntreat
 - Parameters, [66](#)
- NumParticles
 - Event, [8](#)
- Orient
 - GamGamLL, [21](#)
- P
 - Particle, [71–73](#)
- p1mod
 - Parameters, [66](#)
- p2mod
 - Parameters, [66](#)
- P4
 - Particle, [74](#)
- PPtoLL, [76](#)
 - FillKinematics, [79](#)
 - GetEvent, [79](#)
 - IsKinematicsDefined, [79](#)

- SetIncomingParticles, [80](#)
 - SetKinematics, [80](#)
 - SetOutgoingParticles, [81](#)
 - SetPoint, [81](#)
- pair
 - Parameters, [66](#)
- Parameters, [62](#)
 - debug, [65](#)
 - maxmx, [65](#)
 - maxpt, [66](#)
 - mcut, [66](#)
 - minmx, [66](#)
 - minpt, [66](#)
 - ntreat, [66](#)
 - p1mod, [66](#)
 - p2mod, [66](#)
 - pair, [66](#)
 - ReadConfigFile, [65](#)
 - SetEtaRange, [65](#)
 - StoreConfigFile, [65](#)
- Particle, [67](#)
 - _p4, [75](#)
 - AddDaughter, [69](#)
 - Dump, [70](#)
 - E, [70](#)
 - Eta, [70](#)
 - GetDaughters, [70](#)
 - GetLHEline, [71](#)
 - GetMothersIds, [71](#)
 - Hadronise, [71](#)
 - helicity, [75](#)
 - LorentzBoost, [71](#)
 - M, [71](#)
 - P, [71–73](#)
 - P4, [74](#)
 - pdgId, [75](#)
 - Rapidity, [74](#)
 - SetMother, [75](#)
 - status, [75](#)
- pdgId
 - Particle, [75](#)
- PeriPP
 - GamGamLL, [21](#)
- PhysicsBoundaries, [76](#)
- Pickin
 - GamGamLL, [22](#)
- PrepareHadronisation
 - GamGamLL, [22](#)
- Process, [82](#)
 - FillKinematics, [84](#)
 - GetEvent, [85](#)
 - IsKinematicsDefined, [85](#)
 - SetIncomingParticles, [85](#)
 - SetKinematics, [86](#)
 - SetOutgoingParticles, [86](#)
 - SetPoint, [86](#)
- pyjoin
 - Pythia6Hadroniser, [90](#)
- Pythia6Hadroniser, [88](#)
 - GetHadrons, [90](#)
 - Hadronise, [90](#)
 - pyjoin, [90](#)
- Pythia8Hadroniser, [91](#)
 - GetHadrons, [92](#)
 - Hadronise, [92, 94](#)
- Rapidity
 - Particle, [74](#)
- ReadConfigFile
 - Parameters, [65](#)
- SetEtaRange
 - Parameters, [65](#)
- SetGen
 - Vegas, [99](#)
- SetIncomingParticles
 - GamGamLL, [22](#)
 - GamGamWW, [28](#)
 - GamPomVMML, [41](#)
 - PPtoLL, [80](#)
 - Process, [85](#)
- SetKinematics
 - GamGamLL, [23](#)
 - GamGamWW, [28](#)
 - GamPomVMML, [41](#)
 - PPtoLL, [80](#)
 - Process, [86](#)
- SetMother
 - Particle, [75](#)
- SetOutgoingParticles
 - GamGamLL, [23](#)
 - GamGamWW, [28](#)
 - GamPomVMML, [42](#)
 - PPtoLL, [81](#)
 - Process, [86](#)
- SetPoint
 - GamGamLL, [24](#)
 - GamGamWW, [29](#)
 - GamPomVMML, [42](#)
 - PPtoLL, [81](#)
 - Process, [86](#)
- status
 - Particle, [75](#)
- Store
 - Event, [8](#)
- StoreConfigFile
 - Parameters, [65](#)
- StoreEvent
 - Vegas, [99](#)
- time_generation
 - Event, [9](#)
- time_total
 - Event, [9](#)
- Timer, [94](#)
 - elapsed, [95](#)
- Treat

- Vegas, [99](#), [101](#)
- ulmass
 - Jetset7Hadroniser, [57](#)
- VMFlux
 - GamPomVMLL, [42](#)
- VXMass
 - GamPomVMLL, [43](#)
- Vegas, [95](#)
 - _f, [101](#)
 - _nTreatCalls, [102](#)
 - _rTreat, [102](#)
 - DumpGrid, [98](#)
 - F, [98](#)
 - Generate, [98](#)
 - GenerateOneEvent, [98](#)
 - Integrate, [98](#)
 - SetGen, [99](#)
 - StoreEvent, [99](#)
 - Treat, [99](#), [101](#)
 - Vegas, [98](#)