Programm Dateien für Atmel Studio

Generated by Doxygen 1.9.1

Chapter 1

Dépot du projet de C++ de Brice PERES et Hélène HASSAN - RENDU FINAL

(Rapport de projet dans le dossier doc/)

1.1 Simulateur de particules en C++

1.1.1 Utilisation du simulateur

1.1.1.1 Choix des contraintes du système

La structure publique **SimulationConstraints** permet de définir un environnement physique (de dimension $n = 1, 2$ ou 3) dans laquelle le système définit pourra évoluer. L'univers de particules est consistué de cellules (en dimension n également donc) auxquelles on peut attribuer une taille à l'initialisation. *Exemple d'utilisation :*

```
# Création d'un cube de 400 unités de côté dans lequel l'univers de particules évoluera
# Les cellules qui composent l'univers sont des cubes de côté 2.5
SimulationConstraints constraintes = SimulationConstraints(Vector<3>(-200.), Vector<3>(200.));
Universe<3> monUnivers(contraintes, 2.5);
```

1.1.1.2 Choix des paramètres physiques du système

La structure publique **SimulationSettings** permet de définir plusieurs paramètres selon l'effet voulu lors de la simulation. La définition de la structure est présente dans la documentation du projet. ∗Exemple d'utilisation : ∗

```
# Création d'une simulation prenant en compte les forces d'intéraction de Lennard-Joes, avec un pas de
      simulation de 0.00005 secondes et un temps total de simulation de 19.5 secondes. La simulation est
      sauvegardée toutes les 2000 itérations.
SimulationSettings settings = SimulationSettings();
settings.lennard_jones_interaction = true;
settings.physics_time_total = 19.5;
settings.physics_time_step = 0.00005;
settings.iter_count_until_save = 2000;
# Simulation
univers.simulate(settings);
```

1.1.1.3 Initialisation des particules

Il est possible d'initialiser les particules du système de deux manières qui requièrent toutes les deux la vitesse, masse, et catégorie (Neutron, Proton, Electron) pour chacune d'entre elles.

La position de chaque particule peut être donnée explicitement à l'initialisation, dans ce cas-là il faudra appeler la méthode ***add*** de la classe **Universe** qui permettra d'ajouter une seule particule à l'univers.

Il est également possible de générer un univers de particules de même nature (même masse, vitesse, catégorie) régulièrement espacées les unes des autres dans un espace de dimension $n = 1$, 2 ou 3 on peut avec la méthode ***add_packed_particles*** de la classe **Universe** qui renvoie le nombre de particules utilisé pour remplir l'espace en fin de fonction. La description de ces deux méthodes de la classe est présente dans la documentation.

Exemple d'utilisation :

```
// Contraintes de simulation
SimulationConstraints constraints = SimulationConstraints(Vector<2>(-50.), Vector<2>(50.));
Universe<2> univers(constraints, 100);
---

// Création d'un univers avec 4 particules dont la position est au préalable définie
univers.add(Vector<2>(0., 0.), Vector<2>(0., 0.), 1., Category::NEUTRON);
univers.add(Vector<2>(0., 1.), Vector<2>(-1., 0.), 3.0e-6, Category::NEUTRON);
univers.add(Vector<2>(0., 5.36), Vector<2>(-0.425, 0.), 9.55e-4, Category::NEUTRON);
univers.add(Vector<2>(34.75, 0.), Vector<2>(0., 0.0296), 1.0e-14, Category::NEUTRON);
---
// Création d'un univers composé d'un rectangle et d'un cube contenant des particules régulièrements
       espacées de la quantité "spacing".
double spacing = pow(2., 1./6.)*.1;
int a = univers.add_packed_particles(Vector<3>(-20., 20., -20.)*spacing+Vector<3>(0., 5., 0.),
       Vector<3>(20., 60., 20.)*spacing+Vector<3>(0., 5., 0.), Vector<3>(0., -10., 0.), 1., NEUTRON,
       Vector<3>(4, 4, 4));
int b = univers.add_packed_particles(Vector<3>(-80., -20., -80.)*spacing, Vector<3>(80., 20., 80.)*spacing,
       Vector<3>(0.), 1., NEUTRON, Vector<3>(16, 4, 16));
```

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 emhash8 Namespace Reference

Classes

- class HashSet

  A cache-friendly hash table with open addressing, linear/quadratic probing and power-of-two capacity.

Variables

- constexpr uint32_t INACTIVE = 0xAAAAAAAA
- constexpr uint32_t END = 0-0x1u
- constexpr uint32_t EAD = 2
- constexpr static float EMH_DEFAULT_LOAD_FACTOR = 0.80f
- constexpr static uint32_t EMH_CACHE_LINE_SIZE = 64

5.1.1 Variable Documentation

5.1.1.1 EAD

```
constexpr uint32_t emhash8::EAD = 2  [constexpr]
```

Definition at line 74 of file emhash_set8.h.

5.1.1.2 EMH_CACHE_LINE_SIZE

```
constexpr static uint32_t emhash8::EMH_CACHE_LINE_SIZE = 64  [static], [constexpr]
```

Definition at line 80 of file emhash_set8.h.

5.1.1.3 EMH_DEFAULT_LOAD_FACTOR

```
constexpr static float emhash8::EMH_DEFAULT_LOAD_FACTOR = 0.80f  [static], [constexpr]
```

Definition at line 77 of file emhash_set8.h.

5.1.1.4 END

```
constexpr uint32_t emhash8::END = 0-0x1u  [constexpr]
```

Definition at line 73 of file emhash_set8.h.

5.1.1.5 INACTIVE

```
constexpr uint32_t emhash8::INACTIVE = 0xAAAAAAAA  [constexpr]
```

Definition at line 72 of file emhash_set8.h.

Chapter 6

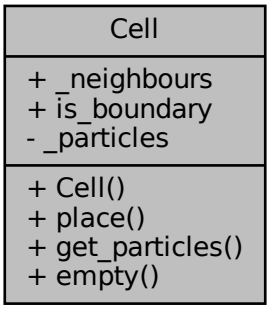
Class Documentation

6.1 Cell Class Reference

Class representing a cell in the grid.

`#include <Cell.h>`

Collaboration diagram for Cell:



Public Member Functions

- Cell ()=default

  Default constructor.
- void place (uint32_t id)

  Place a particle in the cell.
- std::vector< uint32_t > & get_particles ()

  Get the IDs of all particles in the cell.
- void empty ()

  Remove all particles from the cell.

Public Attributes

- std::vector< int32_t > _neighbours

  Indices of neighbouring cells.
- std::vector< bool > is_boundary

  vector of 1+2∗n where n is the dimension of the grid the cell is in. The first value is true if the cell is at least a boundary cell. The second (resp. third) element after is true if the cell is a left (resp. right) boundary for the first dimension. The following couple of values hold the booleans for the next dimensions.

Private Attributes

- std::vector< uint32_t > _particles

  IDs of particles in the cell.

6.1.1 Detailed Description

Class representing a cell in the grid.

Definition at line 9 of file Cell.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Cell()

```
Cell::Cell ( )  [default]
```

Default constructor.

Definition at line 52 of file Cell.h.

6.1.3 Member Function Documentation

6.1.3.1 empty()

```
void Cell::empty ( )
```

Remove all particles from the cell.

Definition at line 48 of file Cell.h.

6.1.3.2 get_particles()

```
std::vector< uint32_t > & Cell::get_particles ( )
```

Get the IDs of all particles in the cell.

Returns

Vector of particle IDs.

Definition at line 44 of file Cell.h.

6.1.3.3 place()

```
void Cell::place (
            uint32_t id )
```

Place a particle in the cell.

Parameters

id	Particle ID.

Definition at line 40 of file Cell.h.

6.1.4 Member Data Documentation

6.1.4.1 _neighbours

```
std::vector<int32_t> Cell::_neighbours
```

Indices of neighbouring cells.

Definition at line 33 of file Cell.h.

6.1.4.2 _particles

```
std::vector<uint32_t> Cell::_particles  [private]
```

IDs of particles in the cell.

Definition at line 36 of file Cell.h.

6.1.4.3 is_boundary

`std::vector<bool> Cell::is_boundary`

vector of 1+2∗n where n is the dimension of the grid the cell is in. The first value is true if the cell is at least a boundary cell. The second (resp. third) element after is true if the cell is a left (resp. right) boundary for the first dimension. The following couple of values hold the booleans for the next dimensions.

Definition at line 34 of file Cell.h.

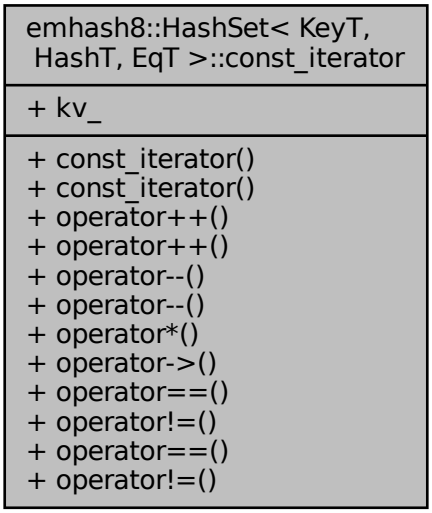
The documentation for this class was generated from the following file:

- include/Cell.h

6.2 emhash8::HashSet< KeyT, HashT, EqT >::const_iterator Class Reference

`#include <emhash_set8.h>`

Collaboration diagram for emhash8::HashSet< KeyT, HashT, EqT >::const_iterator:



Public Types

- using iterator_category = std::bidirectional_iterator_tag
- using value_type = typename htype::value_type
- using difference_type = std::ptrdiff_t
- using pointer = value_type ∗
- using const_pointer = const value_type ∗
- using reference = value_type &
- using const_reference = const value_type &

Public Member Functions

- const_iterator (const iterator &it)
- const_iterator (const htype *hash_map, size_type bucket)
- const_iterator & operator++ ()
- const_iterator operator++ (int)
- const_iterator & operator-- ()
- const_iterator operator-- (int)
- const_reference operator* () const
- const_pointer operator-> () const
- bool operator== (const iterator &rhs) const
- bool operator!= (const iterator &rhs) const
- bool operator== (const const_iterator &rhs) const
- bool operator!= (const const_iterator &rhs) const

Public Attributes

- const value_type * kv_

6.2.1 Detailed Description

template<typename KeyT, typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
class emhash8::HashSet< KeyT, HashT, EqT >::const_iterator

Definition at line 166 of file emhash_set8.h.

6.2.2 Member Typedef Documentation

6.2.2.1 const_pointer

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::const_pointer = const value_type*
```

Definition at line 173 of file emhash_set8.h.

6.2.2.2 const_reference

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::const_reference = const value_type&
```

Definition at line 175 of file emhash_set8.h.

6.2.2.3 difference_type

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::difference_type = std::ptrdiff_t
```

Definition at line 171 of file emhash_set8.h.

6.2.2.4 iterator_category

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::iterator_category = std::bidirectional↩
_iterator_tag
```

Definition at line 169 of file emhash_set8.h.

6.2.2.5 pointer

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::pointer = value_type*
```

Definition at line 172 of file emhash_set8.h.

6.2.2.6 reference

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::reference = value_type&
```

Definition at line 174 of file emhash_set8.h.

6.2.2.7 value_type

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::value_type = typename htype::value_type
```

Definition at line 170 of file emhash_set8.h.

6.2.3 Constructor & Destructor Documentation

6.2.3.1 const_iterator() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::const_iterator (
            const iterator & it )  [inline]
```

Definition at line 177 of file emhash_set8.h.

6.2.3.2 const_iterator() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::const_iterator (
            const htype * hash_map,
            size_type bucket )  [inline]
```

Definition at line 181 of file emhash_set8.h.

6.2.4 Member Function Documentation

6.2.4.1 operator"!=() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
bool emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::operator!= (
            const const_iterator & rhs ) const  [inline]
```

Definition at line 215 of file emhash_set8.h.

6.2.4.2 operator"!=() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
bool emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::operator!= (
            const iterator & rhs ) const  [inline]
```

Definition at line 213 of file emhash_set8.h.

6.2.4.3 operator*()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
const_reference emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::operator* ( ) const
[inline]
```

Definition at line 209 of file emhash_set8.h.

6.2.4.4 operator++() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
const_iterator& emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::operator++ ( )  [inline]
```

Definition at line 185 of file emhash_set8.h.

6.2.4.5 operator++() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
const_iterator emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::operator++ (
            int  )  [inline]
```

Definition at line 191 of file emhash_set8.h.

6.2.4.6 operator--() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
const_iterator& emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::operator-- ( )  [inline]
```

Definition at line 197 of file emhash_set8.h.

6.2.4.7 operator--() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
const_iterator emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::operator-- (
            int  )  [inline]
```

Definition at line 203 of file emhash_set8.h.

6.2.4.8 operator->()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
const_pointer emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::operator-> ( ) const
[inline]
```

Definition at line 210 of file emhash_set8.h.

6.2.4.9 operator==() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
bool emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::operator== (
            const const_iterator & rhs ) const  [inline]
```

Definition at line 214 of file emhash_set8.h.

6.2.4.10 operator==() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
bool emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::operator== (
            const iterator & rhs ) const  [inline]
```

Definition at line 212 of file emhash_set8.h.

6.2.5 Member Data Documentation

6.2.5.1 kv_

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
const value_type* emhash8::HashSet< KeyT, HashT, EqT >::const_iterator::kv_
```

Definition at line 217 of file emhash_set8.h.

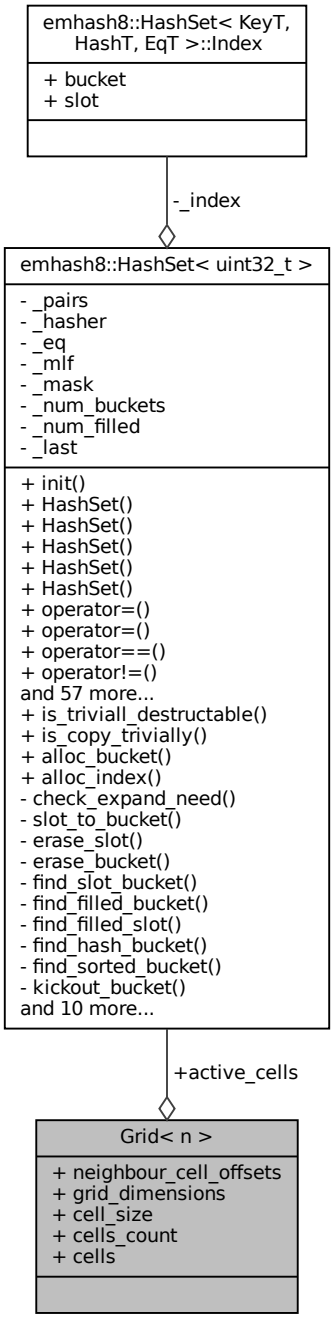
The documentation for this class was generated from the following file:

- include/emhash_set8.h

6.3 Grid< n > Struct Template Reference

`#include <Grid.h>`

Collaboration diagram for Grid< n >:



Public Types

- typedef emhash8::HashSet< uint32_t > CellHashSet

  Fast Hash set used to store active cells.

Public Attributes

- std::array< int, n > neighbour_cell_offsets

  The offset to get the neighbor of a cell for each dimension.
- std::array< int32_t, n > grid_dimensions

  The dimensions of the cells' grid.
- double cell_size

  The physical size of each grid cell.
- uint32_t cells_count

  The number of cells.
- std::vector< Cell > cells

  The array holding the cells.
- CellHashSet active_cells

  The cells currently holding at least one particle.

6.3.1 Detailed Description

template<unsigned int n>
struct Grid< n >

Definition at line 9 of file Grid.h.

6.3.2 Member Typedef Documentation

6.3.2.1 CellHashSet

```
template<unsigned int n>
typedef emhash8::HashSet<uint32_t> Grid< n >::CellHashSet
```

Fast Hash set used to store active cells.

Definition at line 11 of file Grid.h.

6.3.3 Member Data Documentation

6.3.3.1 active_cells

```
template<unsigned int n>
CellHashSet Grid< n >::active_cells
```

The cells currently holding at least one particle.

Definition at line 21 of file Grid.h.

6.3.3.2 cell_size

```
template<unsigned int n>
double Grid< n >::cell_size
```

The physical size of each grid cell.

Definition at line 15 of file Grid.h.

6.3.3.3 cells

```
template<unsigned int n>
std::vector<Cell> Grid< n >::cells
```

The array holding the cells.

Definition at line 20 of file Grid.h.

6.3.3.4 cells_count

```
template<unsigned int n>
uint32_t Grid< n >::cells_count
```

The number of cells.

Definition at line 16 of file Grid.h.

6.3.3.5 grid_dimensions

```
template<unsigned int n>
std::array<int32_t, n> Grid< n >::grid_dimensions
```

The dimensions of the cells' grid.

Definition at line 14 of file Grid.h.

6.3.3.6 neighbour_cell_offsets

```
template<unsigned int n>
std::array<int, n> Grid< n >::neighbour_cell_offsets
```

The offset to get the neighbor of a cell for each dimension.

Definition at line 13 of file Grid.h.

The documentation for this struct was generated from the following file:

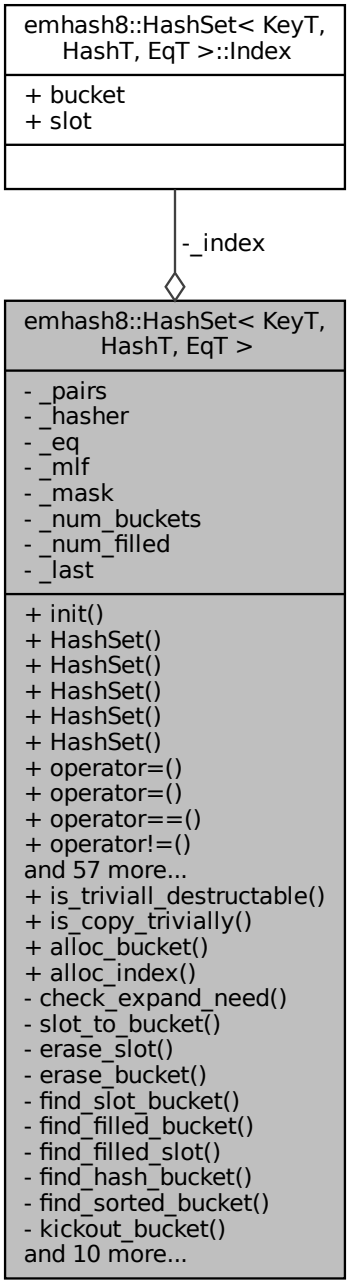
- include/Grid.h

6.4 emhash8::HashSet< KeyT, HashT, EqT > Class Template Reference

A cache-friendly hash table with open addressing, linear/quadratic probing and power-of-two capacity.

`#include <emhash_set8.h>`

Collaboration diagram for emhash8::HashSet< KeyT, HashT, EqT >:



Classes

- class const_iterator
- struct Index
- class iterator

Public Types

- using htype = HashSet< KeyT, HashT, EqT >
- using value_type = KeyT
- using key_type = KeyT
- using size_type = uint32_t
- using hasher = HashT
- using key_equal = EqT

Public Member Functions

- void init (size_type bucket, float mlf=EMH_DEFAULT_LOAD_FACTOR)
- HashSet (size_type bucket=2, float mlf=EMH_DEFAULT_LOAD_FACTOR)
- HashSet (const HashSet &rhs)
- HashSet (HashSet &&rhs)
- HashSet (std::initializer_list< value_type > ilist)
- template<class InputIt >
  HashSet (InputIt first, InputIt last, size_type bucket_count=4)
- HashSet & operator= (const HashSet &rhs)
- HashSet & operator= (HashSet &&rhs)
- template<typename Con >
  bool operator== (const Con &rhs) const
- template<typename Con >
  bool operator!= (const Con &rhs) const
- ~HashSet ()
- void clone (const HashSet &rhs)
- void swap (HashSet &rhs)
- iterator first () const
- iterator last () const
- iterator begin ()
- const_iterator cbegin () const
- const_iterator begin () const
- iterator end ()
- const_iterator cend () const
- const_iterator end () const
- size_type size () const
- bool empty () const
- size_type bucket_count () const
- float load_factor () const

  Returns average number of elements per bucket.
- HashT & hash_function () const
- EqT & key_eq () const
- void max_load_factor (float mlf)
- constexpr float max_load_factor () const
- constexpr size_type max_size () const
- constexpr size_type max_bucket_count () const

- template<typename K = KeyT>
  iterator find (const KeyT &key) noexcept
- template<typename K = KeyT>
  const_iterator find (const K &key) const noexcept
- template<typename K = KeyT>
  bool contains (const K &key) const noexcept
- template<typename K = KeyT>
  size_type count (const K &key) const noexcept
- template<typename K = KeyT>
  std::pair< iterator, iterator > equal_range (const K &key)
- void merge (HashSet &rhs)
- std::pair< iterator, bool > do_insert (const value_type &value)
- std::pair< iterator, bool > do_insert (value_type &&value)
- template<typename K >
  std::pair< iterator, bool > do_insert (K &&key)
- template<typename K >
  std::pair< iterator, bool > do_assign (K &&key)
- std::pair< iterator, bool > insert (const value_type &p)
- std::pair< iterator, bool > insert (value_type &&p)
- void insert (std::initializer_list< value_type > ilist)
- template<typename Iter >
  void insert (Iter first, Iter last)
- template<typename Iter >
  void insert_unique (Iter begin, Iter end)
- template<typename K >
  size_type insert_unique (K &&key)
- size_type insert_unique (value_type &&value)
- size_type insert_unique (const value_type &value)
- template<class... Args>
  std::pair< iterator, bool > emplace (Args &&... args)
- template<class... Args>
  iterator emplace_hint (const_iterator hint, Args &&... args)
- template<class... Args>
  std::pair< iterator, bool > try_emplace (const KeyT &k, Args &&... args)
- template<class... Args>
  std::pair< iterator, bool > try_emplace (KeyT &&k, Args &&... args)
- template<class... Args>
  size_type emplace_unique (Args &&... args)
- std::pair< iterator, bool > insert_or_assign (const KeyT &key)
- std::pair< iterator, bool > insert_or_assign (KeyT &&key)
- size_type erase (const KeyT &key)
- iterator erase (const const_iterator &cit)
- iterator erase (const_iterator first, const_iterator last)
- template<typename Pred >
  size_type erase_if (Pred pred)
- void clearkv ()
- void clear ()

  Remove all elements, keeping full capacity.
- void shrink_to_fit (const float min_factor=EMH_DEFAULT_LOAD_FACTOR/4)
- bool reserve (uint64_t num_elems, bool force)

  Make room for this many elements.
- bool reserve (size_type required_buckets)
- void rebuild (size_type num_buckets)
- void rehash (uint64_t required_buckets)

Static Public Member Functions

- static constexpr bool is_triviall_destructable ()
- static constexpr bool is_copy_trivially ()
- static value_type * alloc_bucket (size_type num_buckets)
- static Index * alloc_index (size_type num_buckets)

Private Member Functions

- bool check_expand_need ()
- size_type slot_to_bucket (const size_type slot) const
- void erase_slot (const size_type sbucket, const size_type main_bucket)
- size_type erase_bucket (const size_type bucket, const size_type main_bucket)
- size_type find_slot_bucket (const size_type slot, size_type &main_bucket) const
- size_type find_filled_bucket (const KeyT &key, uint64_t key_hash) const
- size_type find_filled_slot (const KeyT &key) const
- size_type find_hash_bucket (const KeyT &key) const
- size_type find_sorted_bucket (const KeyT &key) const
- size_type kickout_bucket (const size_type kmain, const size_type bucket)
- size_type find_or_allocate (const KeyT &key, uint64_t key_hash)
- size_type find_unique_bucket (uint64_t key_hash)
- size_type find_empty_bucket (const size_type bucket_from)
- size_type find_last_bucket (size_type main_bucket) const
- size_type find_prev_bucket (const size_type main_bucket, const size_type bucket) const
- size_type hash_bucket (const KeyT &key) const
- size_type hash_main (const size_type bucket) const
- template<typename UType , typename std::enable_if< std::is_integral< UType >::value, uint32_t >::type = 0>
  uint64_t hash_key (const UType key) const
- template<typename UType , typename std::enable_if< std::is_same< UType, std::string >::value, uint32_t >::type = 0>
  uint64_t hash_key (const UType &key) const
- template<typename UType , typename std::enable_if<!std::is_integral< UType >::value &&!std::is_same< UType, std::string >::value, uint32_t >::type = 0>
  uint64_t hash_key (const UType &key) const

Private Attributes

- value_type * _pairs
- Index * _index
- HashT _hasher
- EqT _eq
- uint32_t _mlf
- size_type _mask
- size_type _num_buckets
- size_type _num_filled
- size_type _last

6.4.1 Detailed Description

template<typename KeyT, typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
class emhash8::HashSet< KeyT, HashT, EqT >

A cache-friendly hash table with open addressing, linear/quadratic probing and power-of-two capacity.

Definition at line 85 of file emhash_set8.h.

6.4.2 Member Typedef Documentation

6.4.2.1 hasher

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
using emhash8::HashSet< KeyT, HashT, EqT >::hasher = HashT
```

Definition at line 100 of file emhash_set8.h.

6.4.2.2 htype

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
using emhash8::HashSet< KeyT, HashT, EqT >::htype = HashSet<KeyT, HashT, EqT>
```

Definition at line 88 of file emhash_set8.h.

6.4.2.3 key_equal

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
using emhash8::HashSet< KeyT, HashT, EqT >::key_equal = EqT
```

Definition at line 101 of file emhash_set8.h.

6.4.2.4 key_type

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
using emhash8::HashSet< KeyT, HashT, EqT >::key_type = KeyT
```

Definition at line 90 of file emhash_set8.h.

6.4.2.5 size_type

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
using emhash8::HashSet< KeyT, HashT, EqT >::size_type = uint32_t
```

Definition at line 95 of file emhash_set8.h.

6.4.2.6 value_type

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::value_type = KeyT
```

Definition at line 89 of file emhash_set8.h.

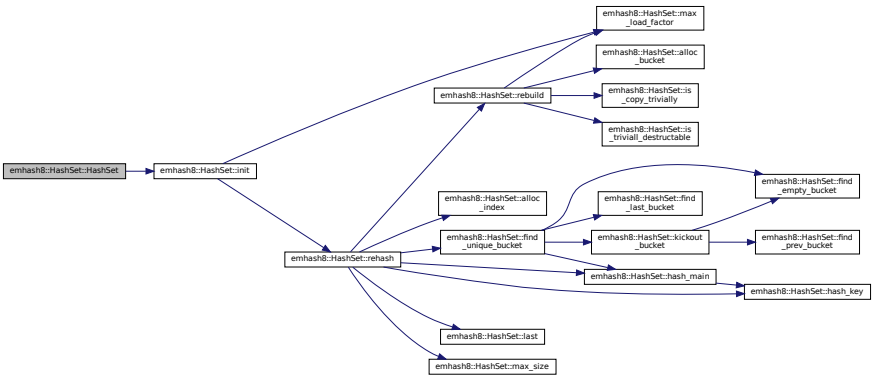
6.4.3 Constructor & Destructor Documentation

6.4.3.1 HashSet() [1/5]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
emhash8::HashSet< KeyT, HashT, EqT >::HashSet (
            size_type bucket = 2,
            float mlf = EMH_DEFAULT_LOAD_FACTOR )  [inline]
```

Definition at line 230 of file emhash_set8.h.

Here is the call graph for this function:

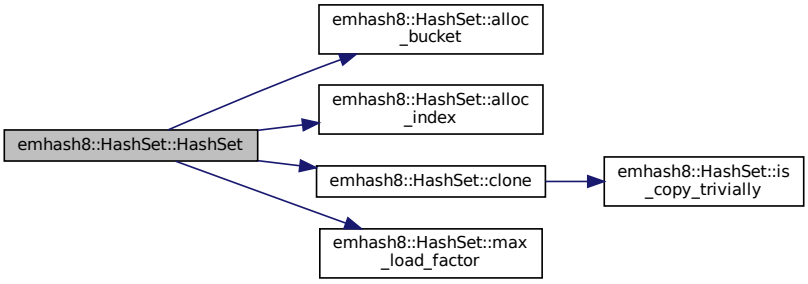


6.4.3.2 HashSet() [2/5]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
emhash8::HashSet< KeyT, HashT, EqT >::HashSet (
            const HashSet< KeyT, HashT, EqT > & rhs )  [inline]
```

Definition at line 235 of file emhash_set8.h.

Here is the call graph for this function:

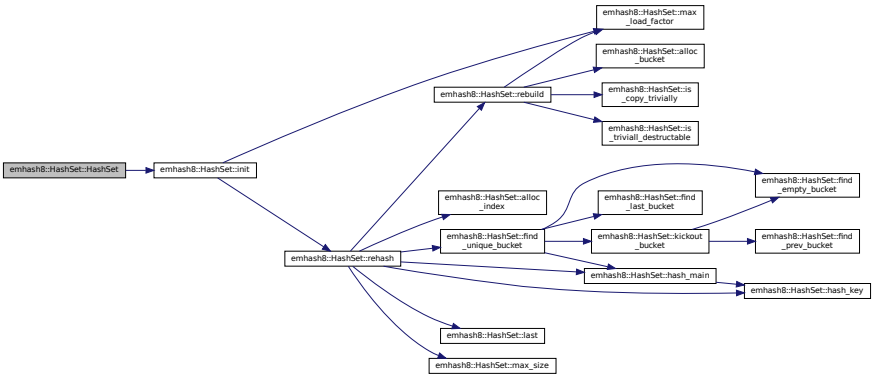


6.4.3.3 HashSet() [3/5]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
emhash8::HashSet< KeyT, HashT, EqT >::HashSet (
            HashSet< KeyT, HashT, EqT > && rhs )  [inline]
```

Definition at line 242 of file emhash_set8.h.

Here is the call graph for this function:

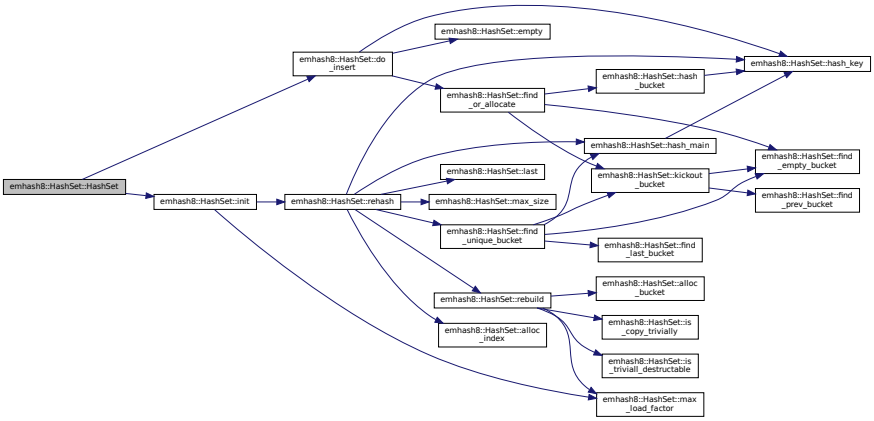


6.4.3.4 HashSet() [4/5]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
emhash8::HashSet< KeyT, HashT, EqT >::HashSet (
            std::initializer_list< value_type > ilist )  [inline]
```

Definition at line 248 of file emhash_set8.h.

Here is the call graph for this function:

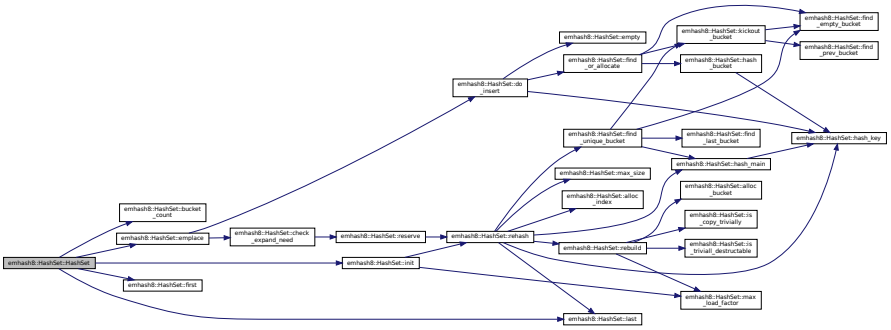


6.4.3.5 HashSet() [5/5]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
template<class InputIt >
emhash8::HashSet< KeyT, HashT, EqT >::HashSet (
            InputIt first,
            InputIt last,
            size_type bucket_count = 4 )  [inline]
```

Definition at line 256 of file emhash_set8.h.

Here is the call graph for this function:



6.4.3.6 ~HashSet()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
emhash8::HashSet< KeyT, HashT, EqT >::~HashSet ( )  [inline]
```

Definition at line 305 of file emhash_set8.h.

Here is the call graph for this function:

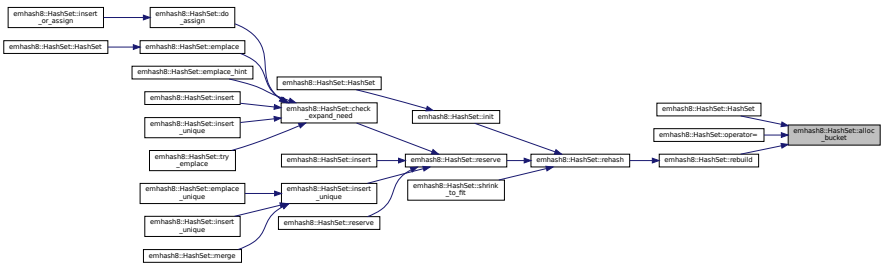
6.4.4 Member Function Documentation

6.4.4.1 alloc_bucket()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
static value_type* emhash8::HashSet< KeyT, HashT, EqT >::alloc_bucket (
            size_type num_buckets )  [inline], [static]
```

Definition at line 827 of file emhash_set8.h.

Here is the caller graph for this function:

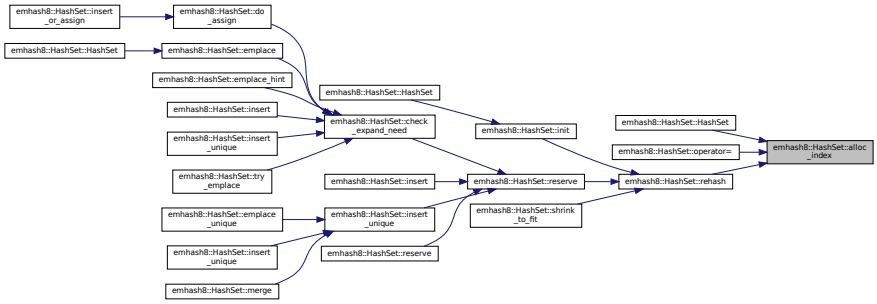


6.4.4.2 alloc_index()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
static Index* emhash8::HashSet< KeyT, HashT, EqT >::alloc_index (
            size_type num_buckets )  [inline], [static]
```

Definition at line 837 of file emhash_set8.h.

Here is the caller graph for this function:



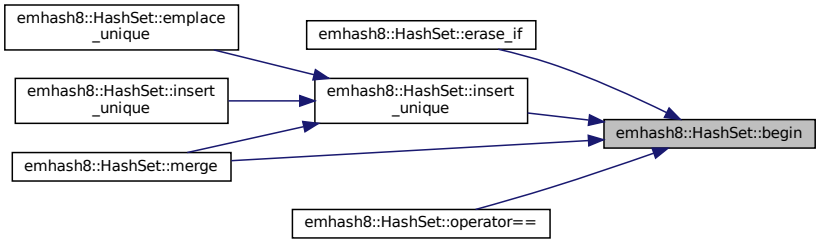
6.4.4.3 begin() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
iterator emhash8::HashSet< KeyT, HashT, EqT >::begin ( )  [inline]
```

Definition at line 351 of file emhash_set8.h.

Here is the call graph for this function:

Here is the caller graph for this function:



6.4.4.4 begin() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
const_iterator emhash8::HashSet< KeyT, HashT, EqT >::begin ( ) const  [inline]
```

Definition at line 353 of file emhash_set8.h.

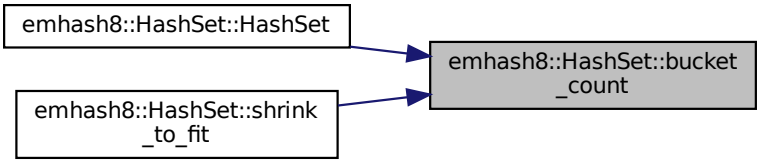
Here is the call graph for this function:

6.4.4.5 bucket_count()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::bucket_count ( ) const  [inline]
```

Definition at line 361 of file emhash_set8.h.

Here is the caller graph for this function:



6.4.4.6 cbegin()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
const_iterator emhash8::HashSet< KeyT, HashT, EqT >::cbegin ( ) const  [inline]
```

Definition at line 352 of file emhash_set8.h.

Here is the call graph for this function:

6.4.4.7 cend()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
const_iterator emhash8::HashSet< KeyT, HashT, EqT >::cend ( ) const  [inline]
```

Definition at line 356 of file emhash_set8.h.

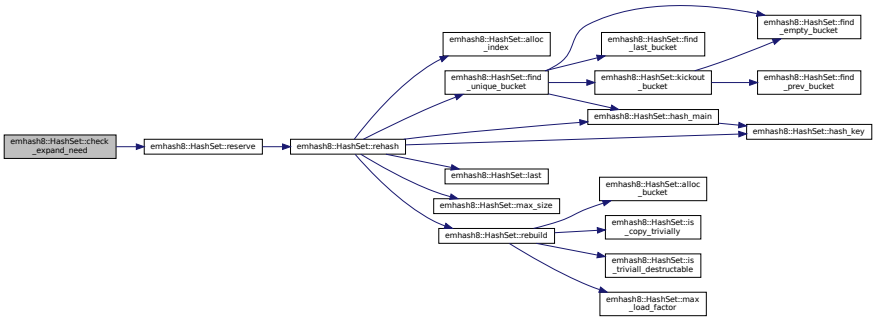
Here is the caller graph for this function:

6.4.4.8 check_expand_need()

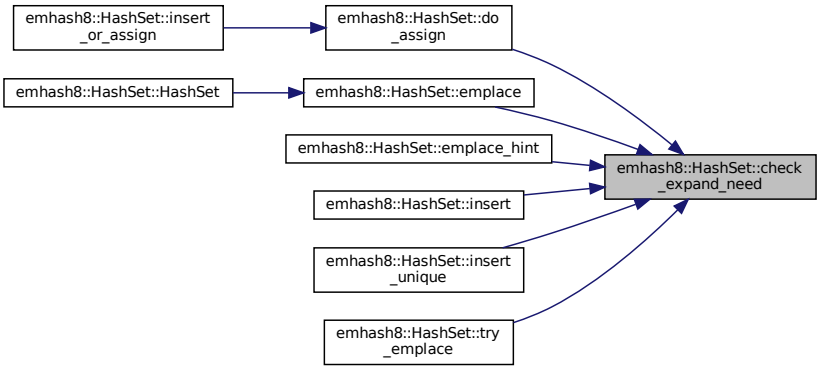
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
bool emhash8::HashSet< KeyT, HashT, EqT >::check_expand_need ( )  [inline], [private]
```

Definition at line 959 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:



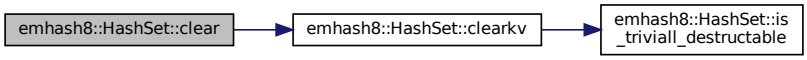
6.4.4.9 clear()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
void emhash8::HashSet< KeyT, HashT, EqT >::clear ( )  [inline]
```

Remove all elements, keeping full capacity.

Definition at line 794 of file emhash_set8.h.

Here is the call graph for this function:

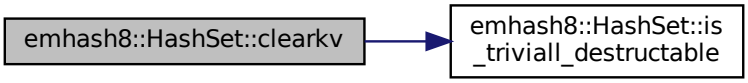


6.4.4.10 clearkv()

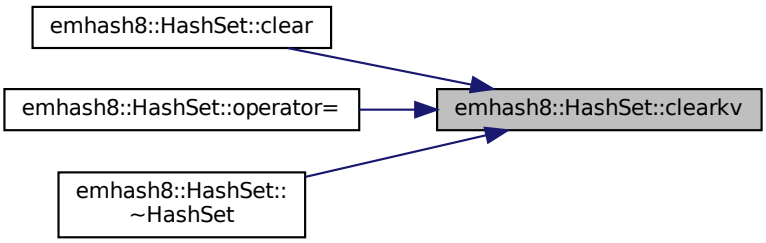
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
void emhash8::HashSet< KeyT, HashT, EqT >::clearkv ( )  [inline]
```

Definition at line 785 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

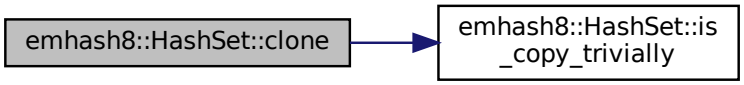


6.4.4.11 clone()

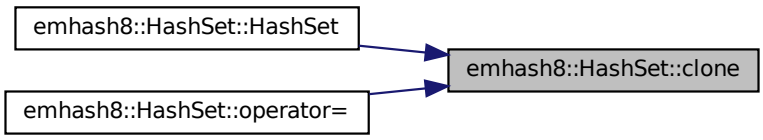
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
void emhash8::HashSet< KeyT, HashT, EqT >::clone (
            const HashSet< KeyT, HashT, EqT > & rhs )  [inline]
```

Definition at line 312 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

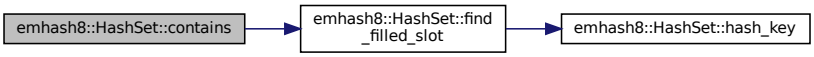


6.4.4.12 contains()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<typename K = KeyT>
bool emhash8::HashSet< KeyT, HashT, EqT >::contains (
            const K & key ) const  [inline], [noexcept]
```

Definition at line 518 of file emhash_set8.h.

Here is the call graph for this function:

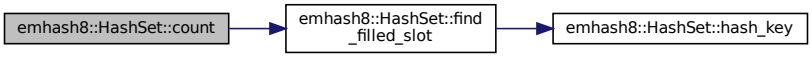


6.4.4.13 count()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<typename K = KeyT>
size_type emhash8::HashSet< KeyT, HashT, EqT >::count (
            const K & key ) const  [inline], [noexcept]
```

Definition at line 524 of file emhash_set8.h.

Here is the call graph for this function:

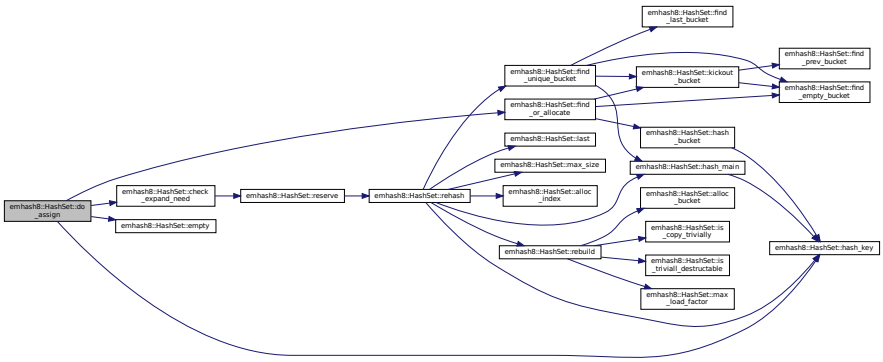


6.4.4.14 do_assign()

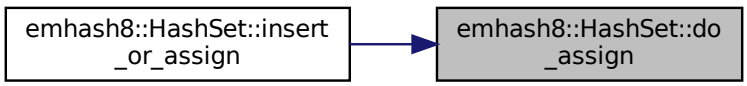
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
template<typename K >
std::pair<iterator, bool> emhash8::HashSet< KeyT, HashT, EqT >::do_assign (
            K && key )  [inline]
```

Definition at line 601 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

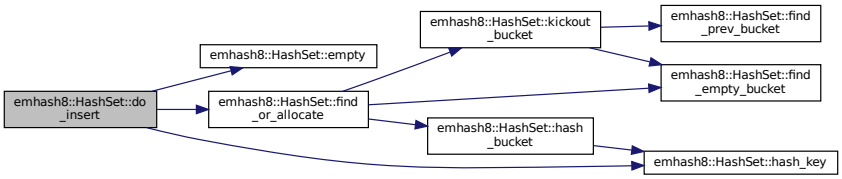


6.4.4.15 do_insert() [1/3]

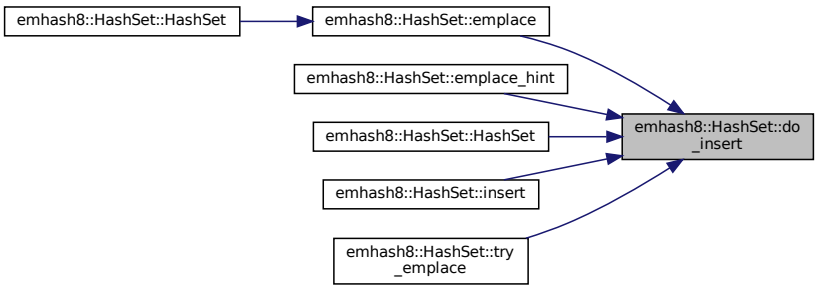
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
std::pair<iterator, bool> emhash8::HashSet< KeyT, HashT, EqT >::do_insert (
            const value_type & value )  [inline]
```

Definition at line 560 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

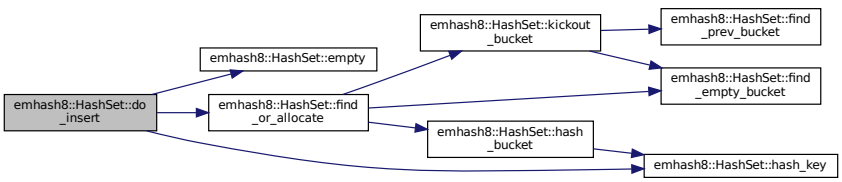


6.4.4.16 do_insert() [2/3]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<typename K >
std::pair<iterator, bool> emhash8::HashSet< KeyT, HashT, EqT >::do_insert (
            K && key )  [inline]
```

Definition at line 587 of file emhash_set8.h.

Here is the call graph for this function:

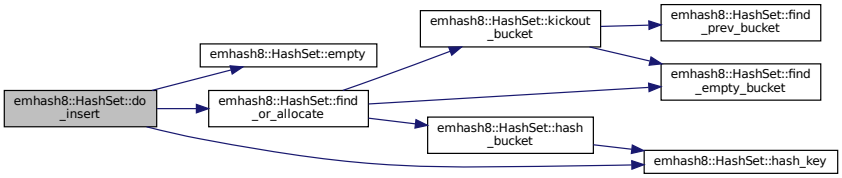


6.4.4.17 do_insert() [3/3]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
std::pair<iterator, bool> emhash8::HashSet< KeyT, HashT, EqT >::do_insert (
            value_type && value )  [inline]
```

Definition at line 573 of file emhash_set8.h.

Here is the call graph for this function:

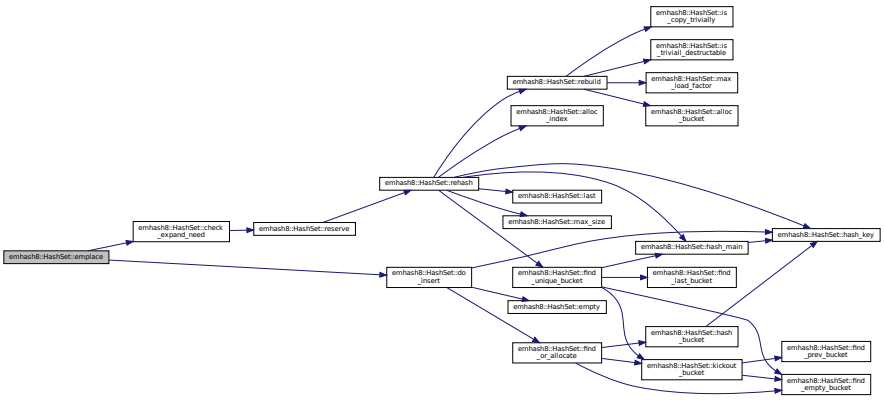


6.4.4.18 emplace()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<class...  Args>
std::pair<iterator, bool> emhash8::HashSet< KeyT, HashT, EqT >::emplace (
            Args &&...  args )  [inline]
```

Definition at line 672 of file emhash_set8.h.

Here is the call graph for this function:



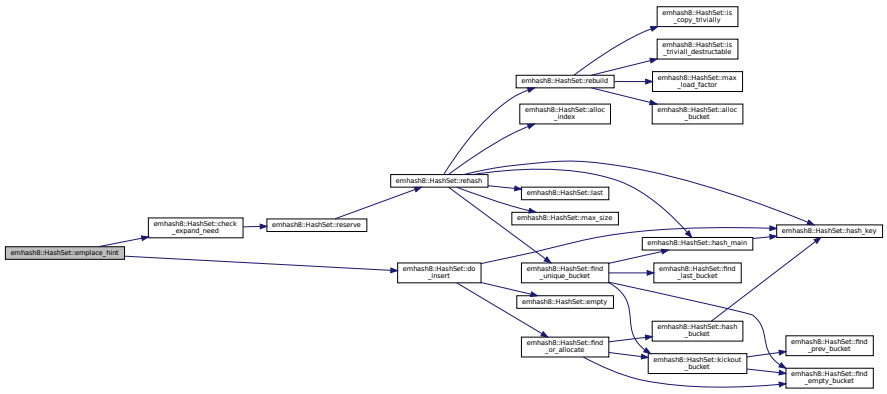
Here is the caller graph for this function:

6.4.4.19 emplace_hint()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<class...  Args>
iterator emhash8::HashSet< KeyT, HashT, EqT >::emplace_hint (
            const_iterator hint,
            Args &&...  args )  [inline]
```

Definition at line 680 of file emhash_set8.h.

Here is the call graph for this function:

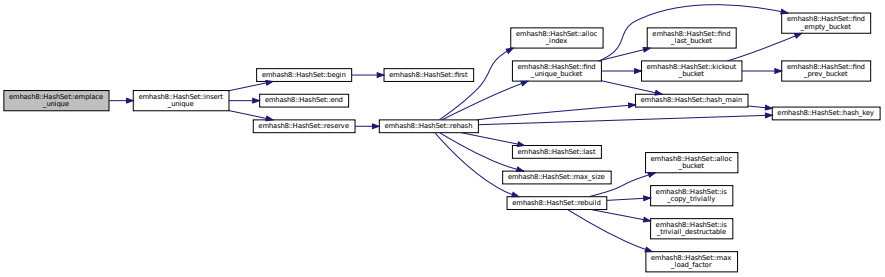


6.4.4.20 emplace_unique()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<class...  Args>
size_type emhash8::HashSet< KeyT, HashT, EqT >::emplace_unique (
            Args &&...  args )  [inline]
```

Definition at line 702 of file emhash_set8.h.

Here is the call graph for this function:

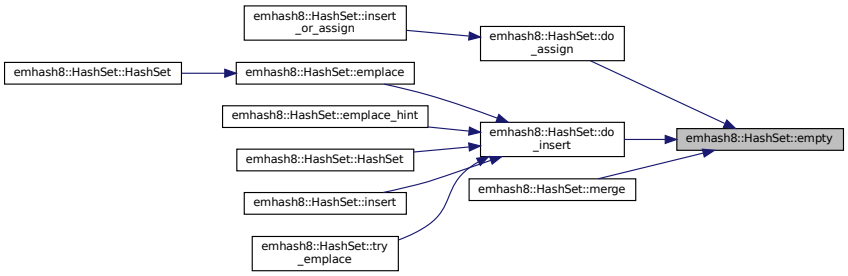


6.4.4.21 empty()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
bool emhash8::HashSet< KeyT, HashT, EqT >::empty ( ) const  [inline]
```

Definition at line 360 of file emhash_set8.h.

Here is the caller graph for this function:

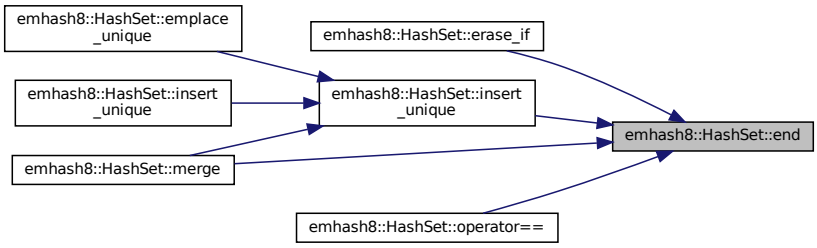


6.4.4.22 end() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
iterator emhash8::HashSet< KeyT, HashT, EqT >::end ( )  [inline]
```

Definition at line 355 of file emhash_set8.h.

Here is the caller graph for this function:



6.4.4.23 end() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
const_iterator emhash8::HashSet< KeyT, HashT, EqT >::end ( ) const  [inline]
```

Definition at line 357 of file emhash_set8.h.

Here is the call graph for this function:

6.4.4.24 equal_range()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
template<typename K = KeyT>
std::pair<iterator, iterator> emhash8::HashSet< KeyT, HashT, EqT >::equal_range (
            const K & key )  [inline]
```

Definition at line 532 of file emhash_set8.h.

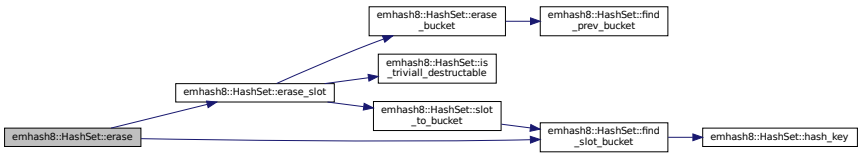
Here is the call graph for this function:

6.4.4.25 erase() [1/3]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
iterator emhash8::HashSet< KeyT, HashT, EqT >::erase (
            const const_iterator & cit )  [inline]
```

Definition at line 725 of file emhash_set8.h.

Here is the call graph for this function:



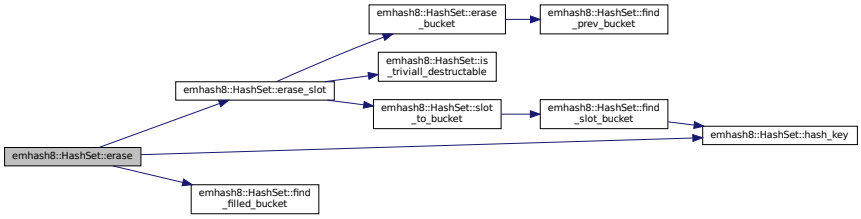
6.4.4.26 erase() [2/3]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::erase (
            const KeyT & key )  [inline]
```

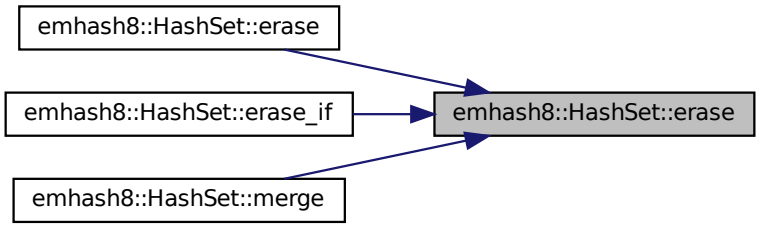
Erase an element from the hash table. return 0 if element was not found

Definition at line 712 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

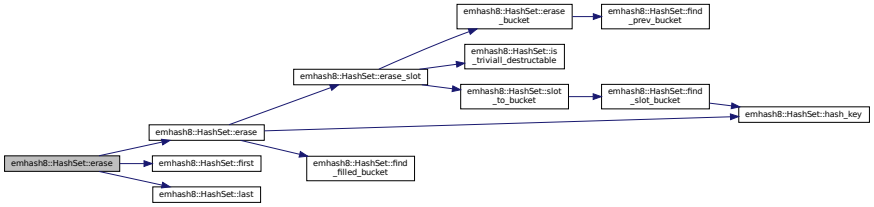


6.4.4.27 erase() [3/3]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
iterator emhash8::HashSet< KeyT, HashT, EqT >::erase (
            const_iterator first,
            const_iterator last )  [inline]
```

Definition at line 735 of file emhash_set8.h.

Here is the call graph for this function:

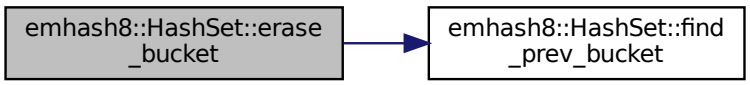


6.4.4.28 erase_bucket()

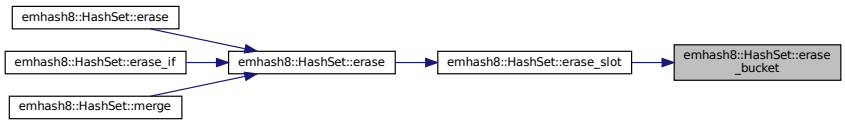
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::erase_bucket (
            const size_type bucket,
            const size_type main_bucket )  [inline], [private]
```

Definition at line 988 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

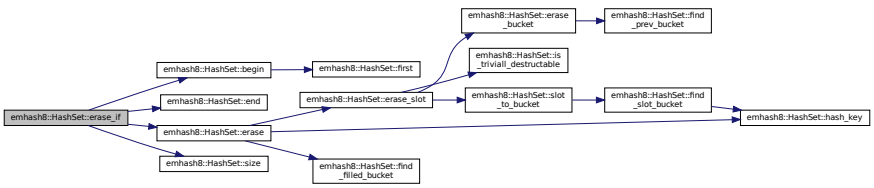


6.4.4.29 erase_if()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<typename Pred >
size_type emhash8::HashSet< KeyT, HashT, EqT >::erase_if (
            Pred pred )  [inline]
```

Definition at line 755 of file emhash_set8.h.

Here is the call graph for this function:

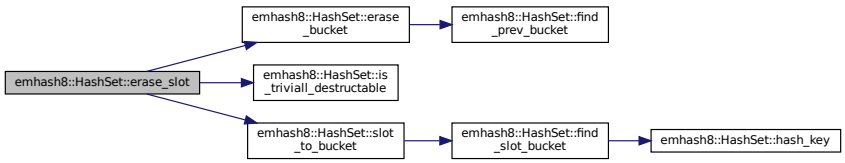


6.4.4.30 erase_slot()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
void emhash8::HashSet< KeyT, HashT, EqT >::erase_slot (
            const size_type sbucket,
            const size_type main_bucket )  [inline], [private]
```

Definition at line 971 of file emhash_set8.h.

Here is the call graph for this function:



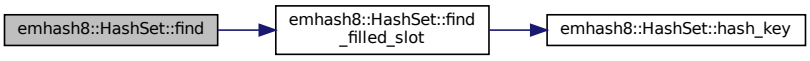
Here is the caller graph for this function:

6.4.4.31 find() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<typename K = KeyT>
const_iterator emhash8::HashSet< KeyT, HashT, EqT >::find (
            const K & key ) const  [inline], [noexcept]
```

Definition at line 512 of file emhash_set8.h.

Here is the call graph for this function:

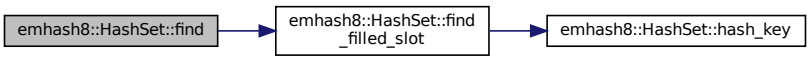


6.4.4.32 find() [2/2]

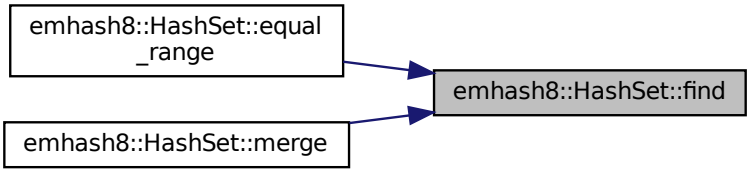
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
template<typename K = KeyT>
iterator emhash8::HashSet< KeyT, HashT, EqT >::find (
            const KeyT & key )  [inline], [noexcept]
```

Definition at line 506 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

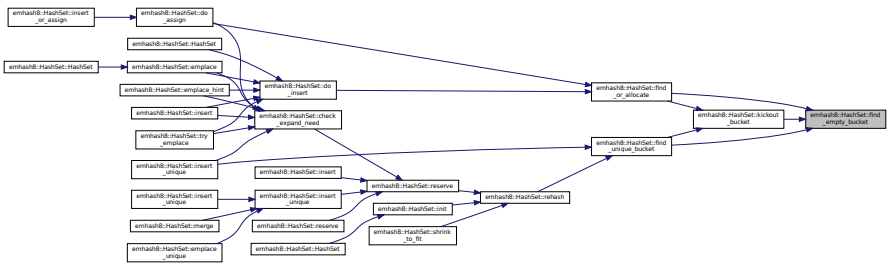


6.4.4.33 find_empty_bucket()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::find_empty_bucket (
            const size_type bucket_from )  [inline], [private]
```

Definition at line 1258 of file emhash_set8.h.

Here is the caller graph for this function:

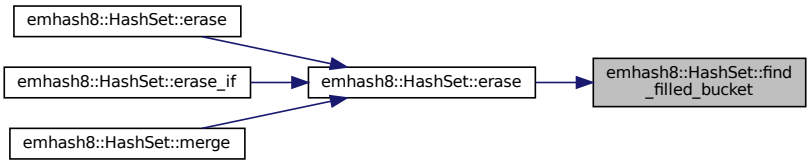


6.4.4.34 find_filled_bucket()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::find_filled_bucket (
            const KeyT & key,
            uint64_t key_hash ) const  [inline], [private]
```

Definition at line 1028 of file emhash_set8.h.

Here is the caller graph for this function:

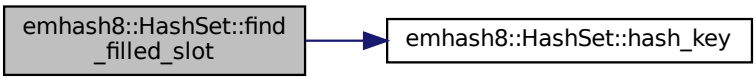


6.4.4.35 find_filled_slot()

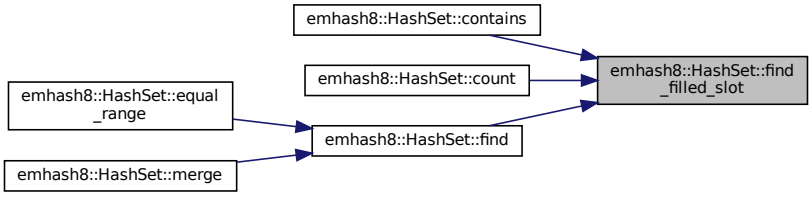
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::find_filled_slot (
            const KeyT & key ) const  [inline], [private]
```

Definition at line 1059 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

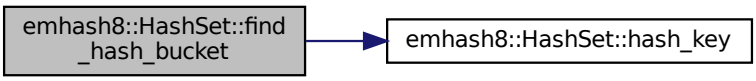


6.4.4.36 find_hash_bucket()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::find_hash_bucket (
            const KeyT & key ) const  [inline], [private]
```

Definition at line 1091 of file emhash_set8.h.

Here is the call graph for this function:

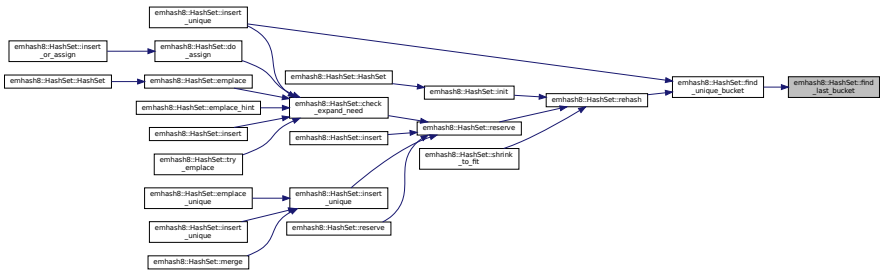


6.4.4.37 find_last_bucket()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::find_last_bucket (
            size_type main_bucket ) const  [inline], [private]
```

Definition at line 1319 of file emhash_set8.h.

Here is the caller graph for this function:

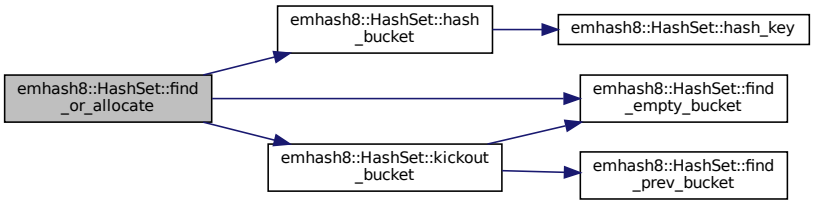


6.4.4.38 find_or_allocate()

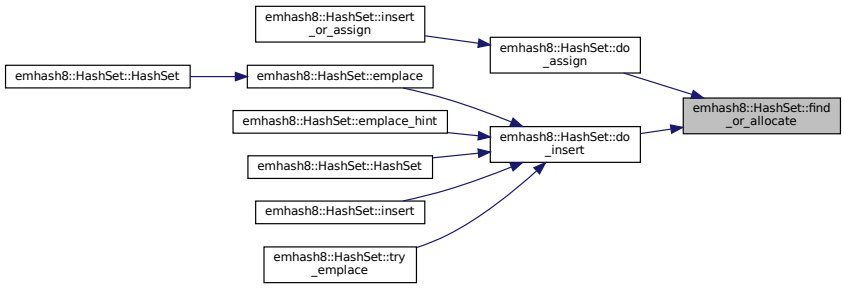
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::find_or_allocate (
            const KeyT & key,
            uint64_t key_hash )  [inline], [private]
```

Definition at line 1186 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

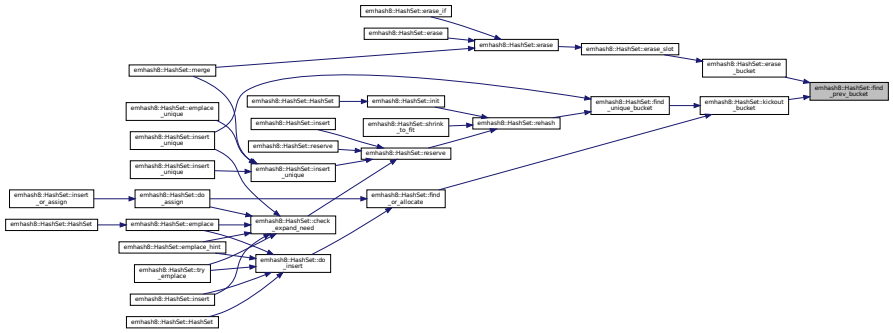


6.4.4.39 find_prev_bucket()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::find_prev_bucket (
            const size_type main_bucket,
            const size_type bucket ) const  [inline], [private]
```

Definition at line 1333 of file emhash_set8.h.

Here is the caller graph for this function:

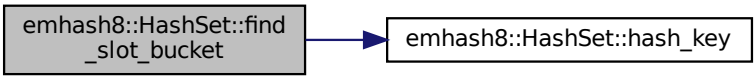


6.4.4.40 find_slot_bucket()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::find_slot_bucket (
            const size_type slot,
            size_type & main_bucket ) const  [inline], [private]
```

Definition at line 1008 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

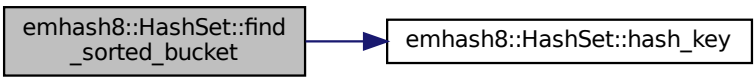
6.4.4.41 find_sorted_bucket()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::find_sorted_bucket (
            const KeyT & key ) const  [inline], [private]
```

|| key < EMH_KEY(_pairs, slot)

Definition at line 1122 of file emhash_set8.h.

Here is the call graph for this function:

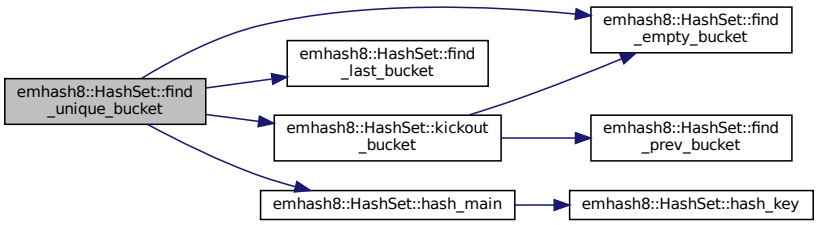


6.4.4.42 find_unique_bucket()

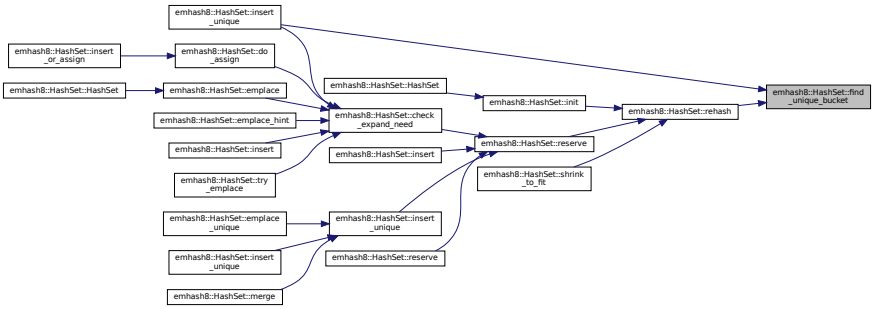
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::find_unique_bucket (
            uint64_t key_hash )  [inline], [private]
```

Definition at line 1225 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

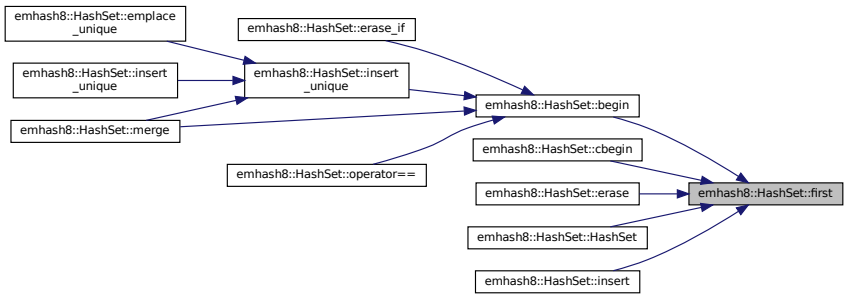


6.4.4.43 first()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
iterator emhash8::HashSet< KeyT, HashT, EqT >::first ( ) const  [inline]
```

Definition at line 348 of file emhash_set8.h.

Here is the caller graph for this function:

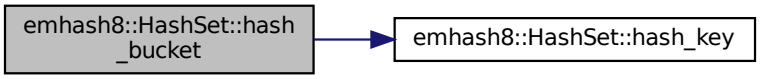


6.4.4.44 hash_bucket()

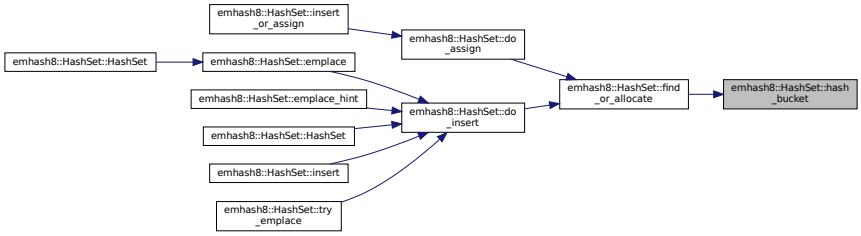
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::hash_bucket (
            const KeyT & key ) const  [inline], [private]
```

Definition at line 1347 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:



6.4.4.45 hash_function()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
HashT& emhash8::HashSet< KeyT, HashT, EqT >::hash_function ( ) const  [inline]
```

Definition at line 366 of file emhash_set8.h.

6.4.4.46 hash_key() [1/3]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<typename UType , typename std::enable_if< std::is_same< UType, std::string >::value,
uint32_t >::type = 0>
uint64_t emhash8::HashSet< KeyT, HashT, EqT >::hash_key (
            const UType & key ) const  [inline], [private]
```

Definition at line 1491 of file emhash_set8.h.

6.4.4.47 hash_key() [2/3]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
template<typename UType , typename std::enable_if<!std::is_integral< UType >::value &&!std::is_same< UType, std::string >::value, uint32_t >::type = 0>
uint64_t emhash8::HashSet< KeyT, HashT, EqT >::hash_key (
            const UType & key ) const  [inline], [private]
```

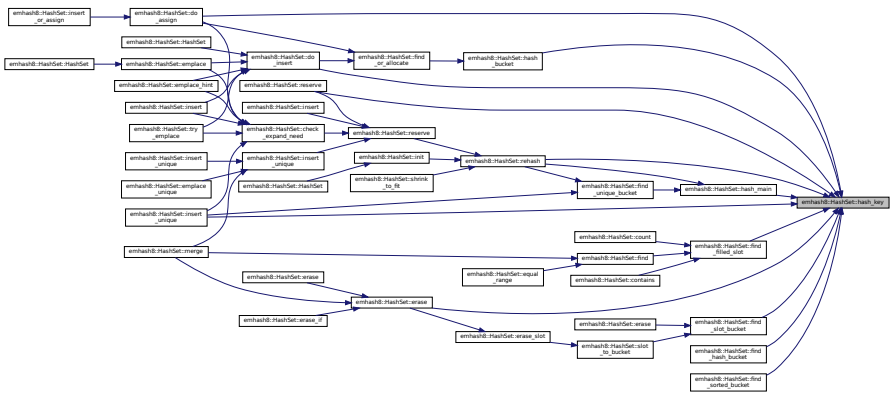
Definition at line 1503 of file emhash_set8.h.

6.4.4.48 hash_key() [3/3]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
template<typename UType , typename std::enable_if< std::is_integral< UType >::value, uint32_t >::type = 0>
uint64_t emhash8::HashSet< KeyT, HashT, EqT >::hash_key (
            const UType key ) const  [inline], [private]
```

Definition at line 1477 of file emhash_set8.h.

Here is the caller graph for this function:



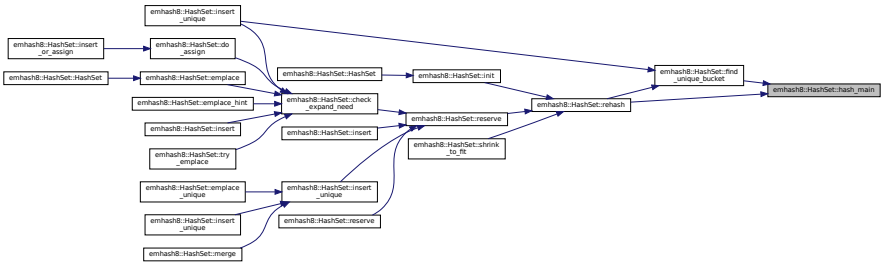
6.4.4.49 hash_main()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::hash_main (
            const size_type bucket ) const  [inline], [private]
```

Definition at line 1352 of file emhash_set8.h.

Here is the call graph for this function:

Here is the caller graph for this function:

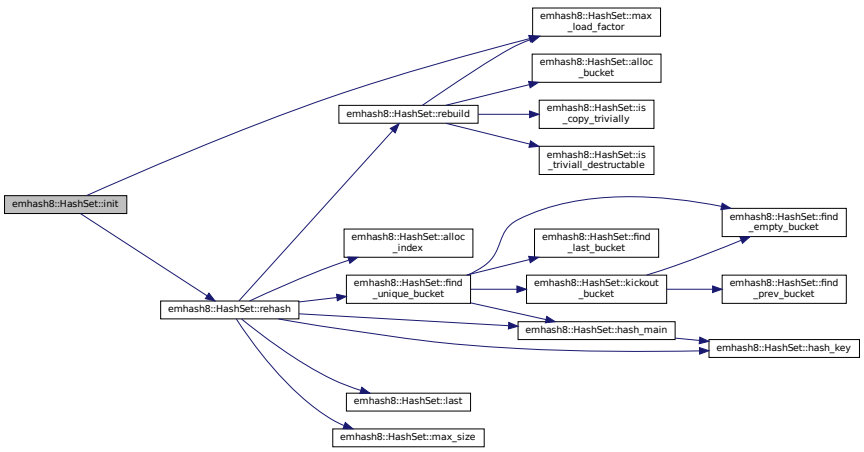


6.4.4.50 init()

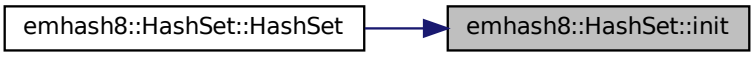
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
void emhash8::HashSet< KeyT, HashT, EqT >::init (
            size_type bucket,
            float mlf = EMH_DEFAULT_LOAD_FACTOR )  [inline]
```

Definition at line 220 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

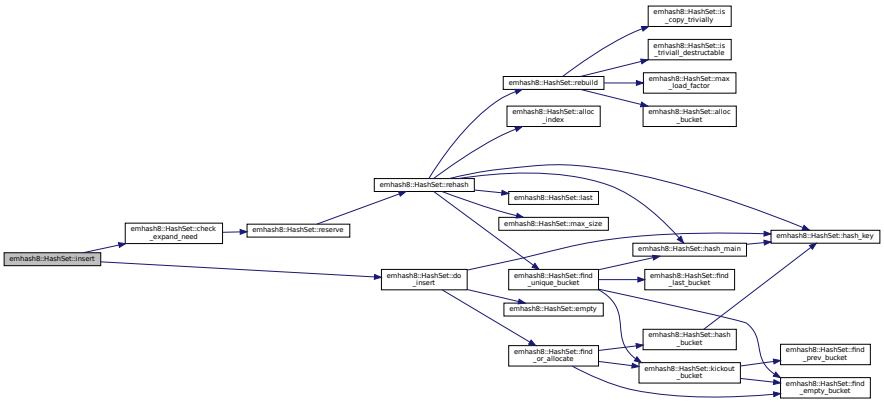


6.4.4.51 insert() [1/4]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
std::pair<iterator, bool> emhash8::HashSet< KeyT, HashT, EqT >::insert (
            const value_type & p )  [inline]
```

Definition at line 615 of file emhash_set8.h.

Here is the call graph for this function:

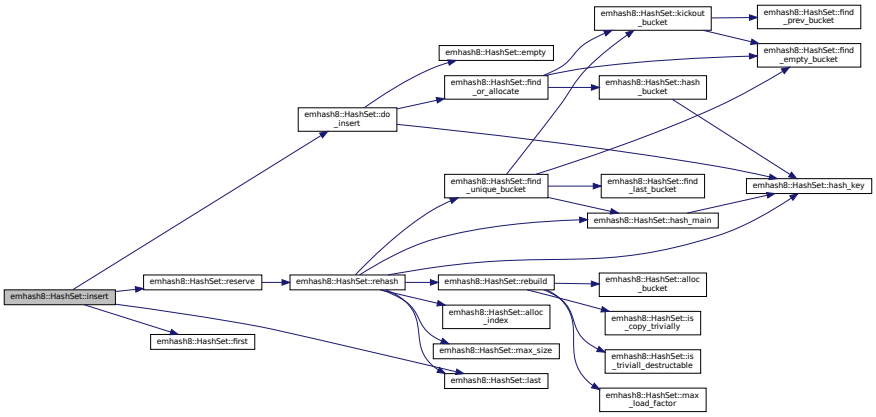


6.4.4.52 insert() [2/4]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<typename Iter >
void emhash8::HashSet< KeyT, HashT, EqT >::insert (
            Iter first,
            Iter last )  [inline]
```

Definition at line 635 of file emhash_set8.h.

Here is the call graph for this function:

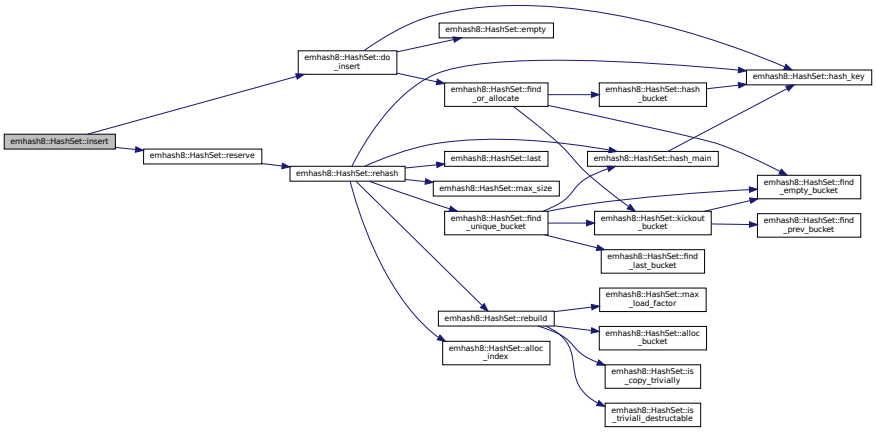


6.4.4.53 insert() [3/4]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
void emhash8::HashSet< KeyT, HashT, EqT >::insert (
            std::initializer_list< value_type > ilist )  [inline]
```

Definition at line 627 of file emhash_set8.h.

Here is the call graph for this function:

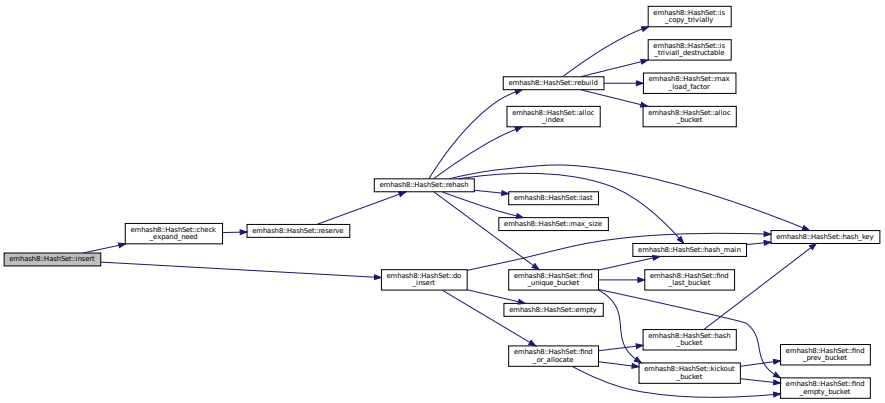


6.4.4.54 insert() [4/4]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
std::pair<iterator, bool> emhash8::HashSet< KeyT, HashT, EqT >::insert (
            value_type && p )  [inline]
```

Definition at line 621 of file emhash_set8.h.

Here is the call graph for this function:

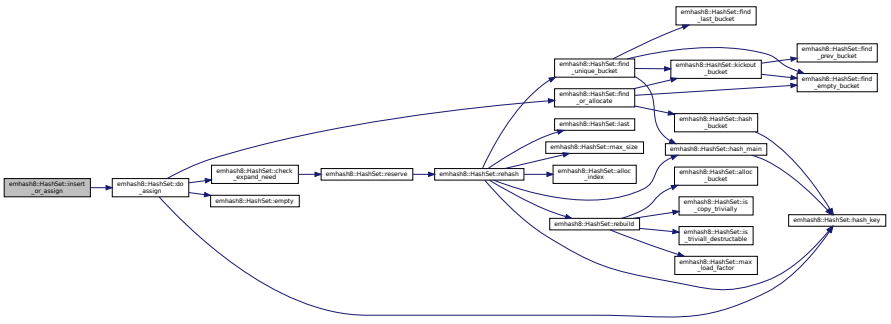


6.4.4.55 insert_or_assign() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
std::pair<iterator, bool> emhash8::HashSet< KeyT, HashT, EqT >::insert_or_assign (
            const KeyT & key )  [inline]
```

Definition at line 707 of file emhash_set8.h.

Here is the call graph for this function:

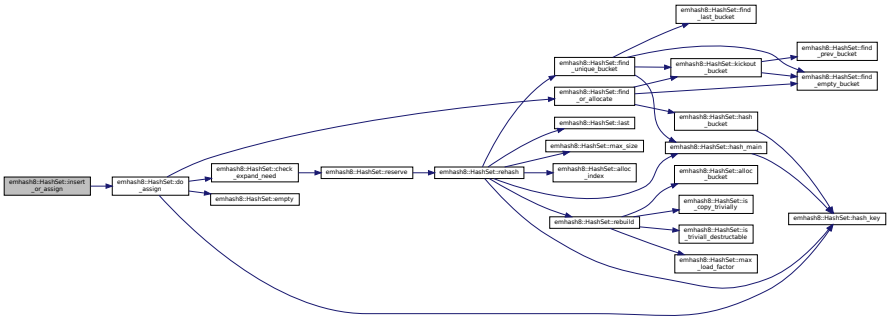


6.4.4.56 insert_or_assign() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
std::pair<iterator, bool> emhash8::HashSet< KeyT, HashT, EqT >::insert_or_assign (
            KeyT && key )  [inline]
```

Definition at line 708 of file emhash_set8.h.

Here is the call graph for this function:

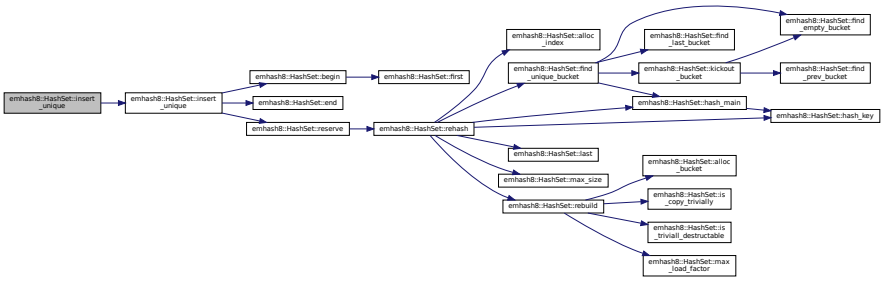


6.4.4.57 insert_unique() [1/4]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::insert_unique (
            const value_type & value )  [inline]
```

Definition at line 666 of file emhash_set8.h.

Here is the call graph for this function:

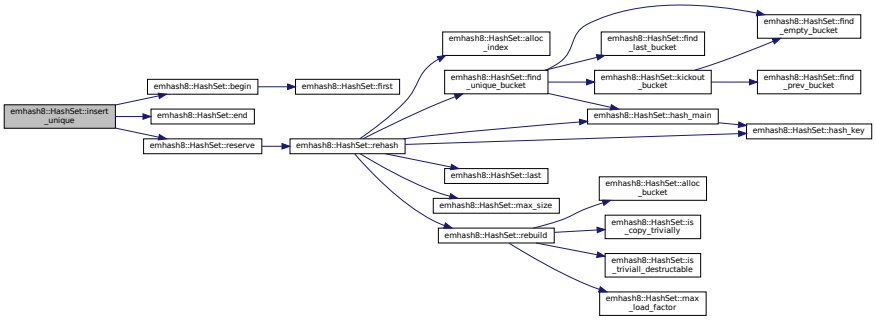


6.4.4.58 insert_unique() [2/4]

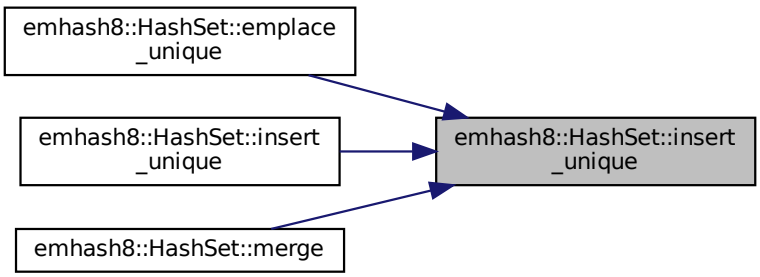
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<typename Iter >
void emhash8::HashSet< KeyT, HashT, EqT >::insert_unique (
            Iter begin,
            Iter end )  [inline]
```

Definition at line 643 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

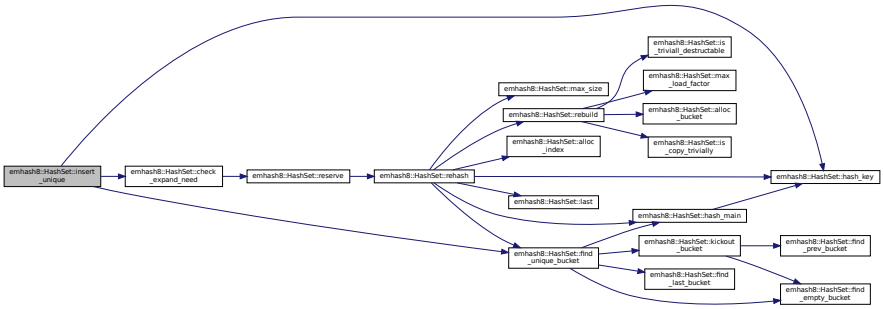


6.4.4.59 insert_unique() [3/4]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<typename K >
size_type emhash8::HashSet< KeyT, HashT, EqT >::insert_unique (
            K && key )  [inline]
```

Definition at line 652 of file emhash_set8.h.

Here is the call graph for this function:

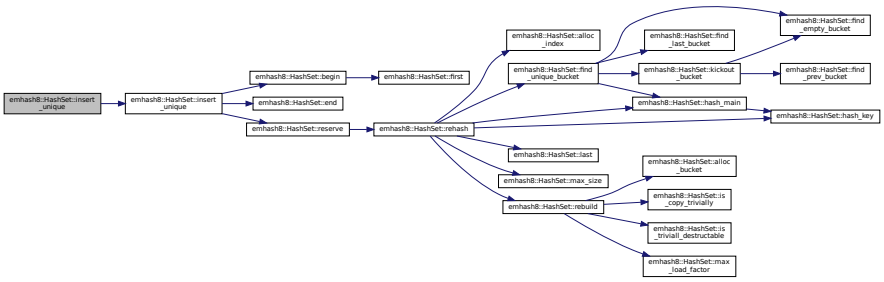


6.4.4.60 insert_unique() [4/4]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::insert_unique (
            value_type && value )  [inline]
```

Definition at line 661 of file emhash_set8.h.

Here is the call graph for this function:

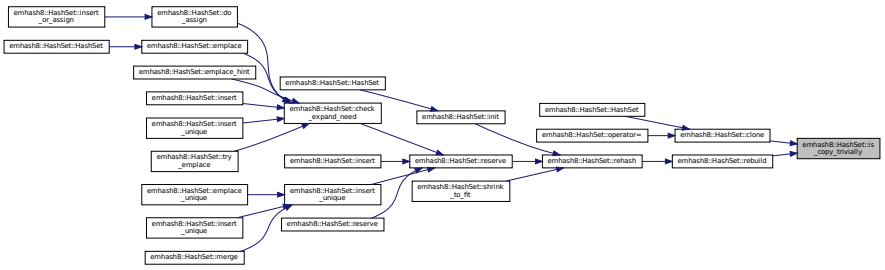


6.4.4.61 is_copy_trivially()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
static constexpr bool emhash8::HashSet< KeyT, HashT, EqT >::is_copy_trivially ( )  [inline], [static], [constexpr]
```

Definition at line 776 of file emhash_set8.h.

Here is the caller graph for this function:

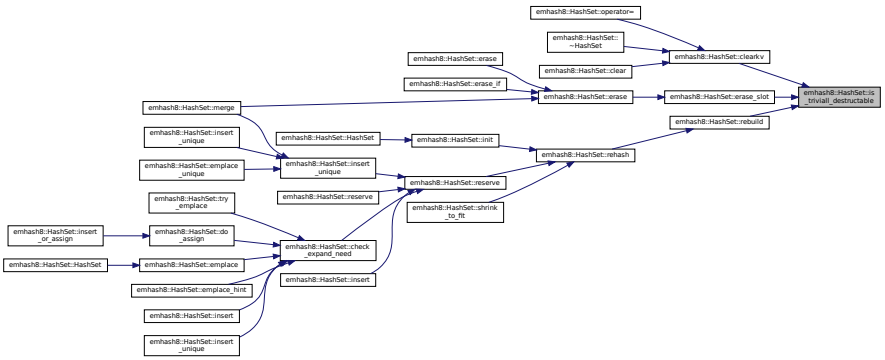


6.4.4.62 is_triviall_destructable()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
static constexpr bool emhash8::HashSet< KeyT, HashT, EqT >::is_triviall_destructable ( )
[inline], [static], [constexpr]
```

Definition at line 767 of file emhash_set8.h.

Here is the caller graph for this function:



6.4.4.63 key_eq()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
EqT& emhash8::HashSet< KeyT, HashT, EqT >::key_eq ( ) const  [inline]
```

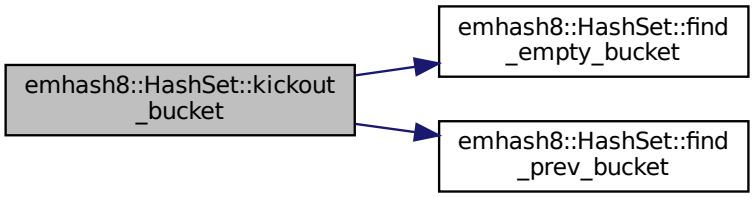
Definition at line 367 of file emhash_set8.h.

6.4.4.64 kickout_bucket()

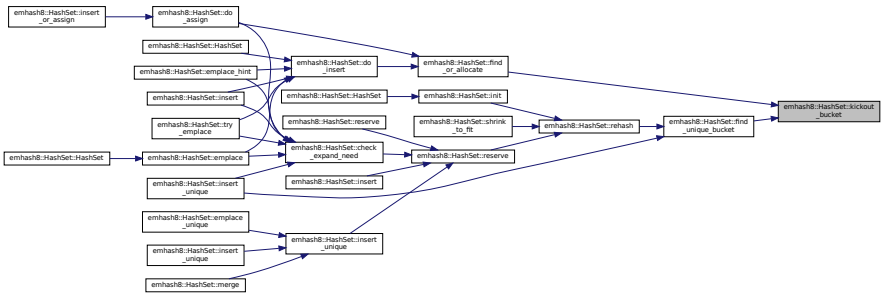
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::kickout_bucket (
            const size_type kmain,
            const size_type bucket )  [inline], [private]
```

Definition at line 1161 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

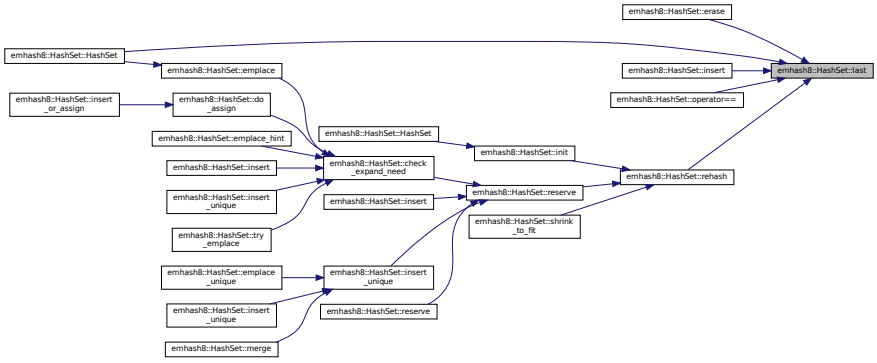


6.4.4.65 last()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
iterator emhash8::HashSet< KeyT, HashT, EqT >::last ( ) const  [inline]
```

Definition at line 349 of file emhash_set8.h.

Here is the caller graph for this function:



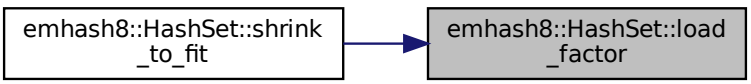
6.4.4.66 load_factor()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
float emhash8::HashSet< KeyT, HashT, EqT >::load_factor ( ) const  [inline]
```

Returns average number of elements per bucket.

Definition at line 364 of file emhash_set8.h.

Here is the caller graph for this function:

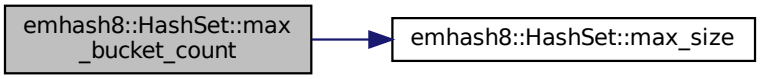


6.4.4.67 max_bucket_count()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
constexpr size_type emhash8::HashSet< KeyT, HashT, EqT >::max_bucket_count ( ) const  [inline],
[constexpr]
```

Definition at line 377 of file emhash_set8.h.

Here is the call graph for this function:

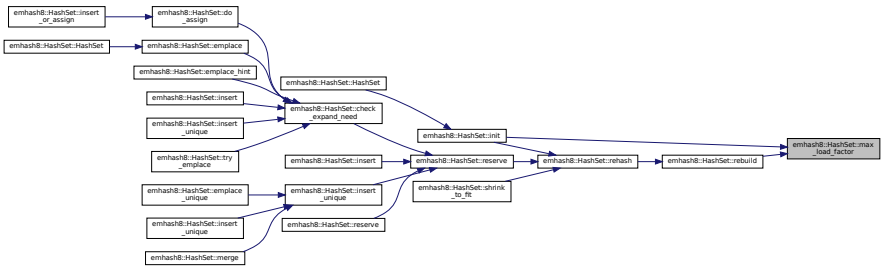


6.4.4.68 max_load_factor() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
constexpr float emhash8::HashSet< KeyT, HashT, EqT >::max_load_factor ( ) const  [inline], [constexpr]
```

Definition at line 375 of file emhash_set8.h.

Here is the caller graph for this function:

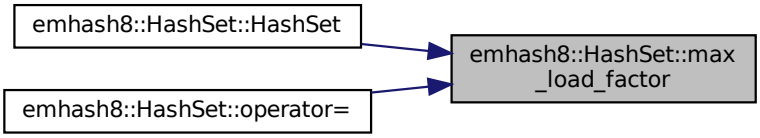


6.4.4.69 max_load_factor() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
void emhash8::HashSet< KeyT, HashT, EqT >::max_load_factor (
            float mlf )  [inline]
```

Definition at line 369 of file emhash_set8.h.

Here is the caller graph for this function:

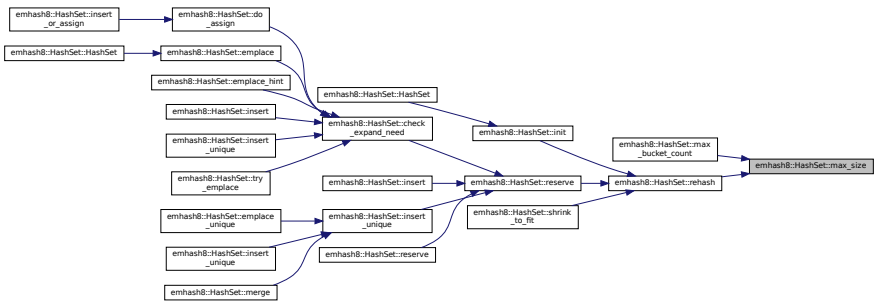


6.4.4.70 max_size()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
constexpr size_type emhash8::HashSet< KeyT, HashT, EqT >::max_size ( ) const  [inline], [constexpr]
```

Definition at line 376 of file emhash_set8.h.

Here is the caller graph for this function:

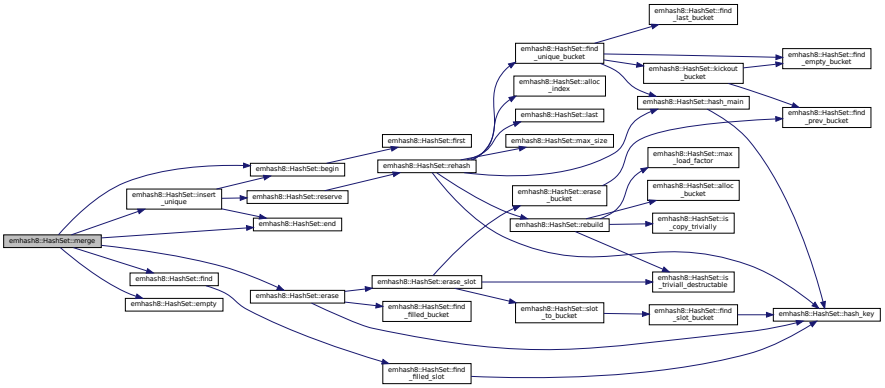


6.4.4.71 merge()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
void emhash8::HashSet< KeyT, HashT, EqT >::merge (
            HashSet< KeyT, HashT, EqT > & rhs )  [inline]
```

Definition at line 541 of file emhash_set8.h.

Here is the call graph for this function:



6.4.4.72 operator"!=()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<typename Con >
bool emhash8::HashSet< KeyT, HashT, EqT >::operator!= (
            const Con & rhs ) const  [inline]
```

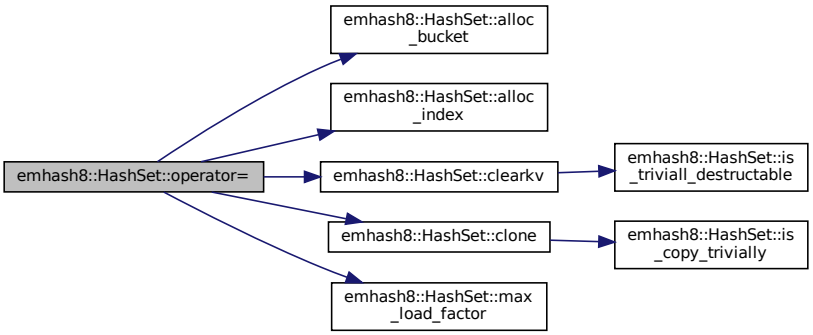
Definition at line 303 of file emhash_set8.h.

6.4.4.73 operator=() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
HashSet& emhash8::HashSet< KeyT, HashT, EqT >::operator= (
            const HashSet< KeyT, HashT, EqT > & rhs )  [inline]
```

Definition at line 263 of file emhash_set8.h.

Here is the call graph for this function:

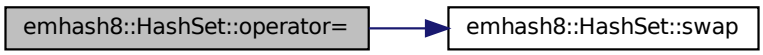


6.4.4.74 operator=() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
HashSet& emhash8::HashSet< KeyT, HashT, EqT >::operator= (
            HashSet< KeyT, HashT, EqT > && rhs )  [inline]
```

Definition at line 279 of file emhash_set8.h.

Here is the call graph for this function:

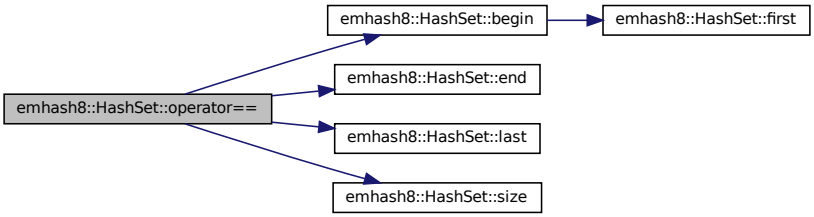


6.4.4.75 operator==()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<typename Con >
bool emhash8::HashSet< KeyT, HashT, EqT >::operator== (
            const Con & rhs ) const  [inline]
```

Definition at line 289 of file emhash_set8.h.

Here is the call graph for this function:

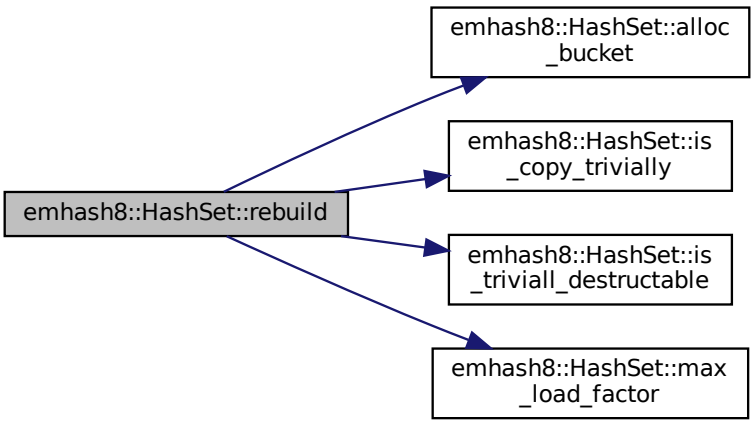


6.4.4.76 rebuild()

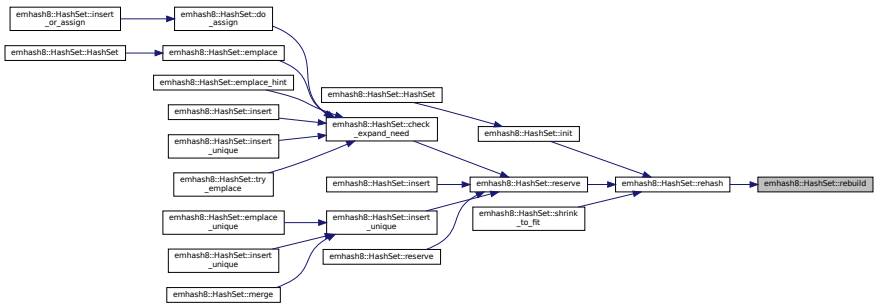
```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
void emhash8::HashSet< KeyT, HashT, EqT >::rebuild (
            size_type num_buckets )  [inline]
```

Definition at line 877 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:



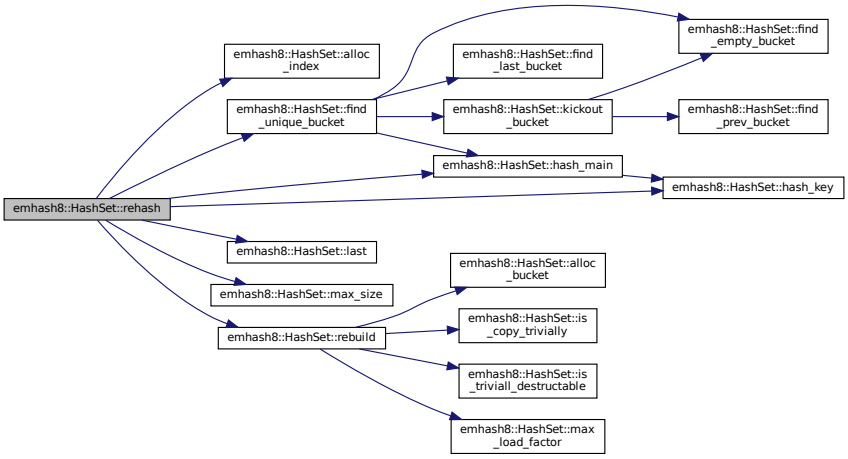
6.4.4.77 rehash()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
```

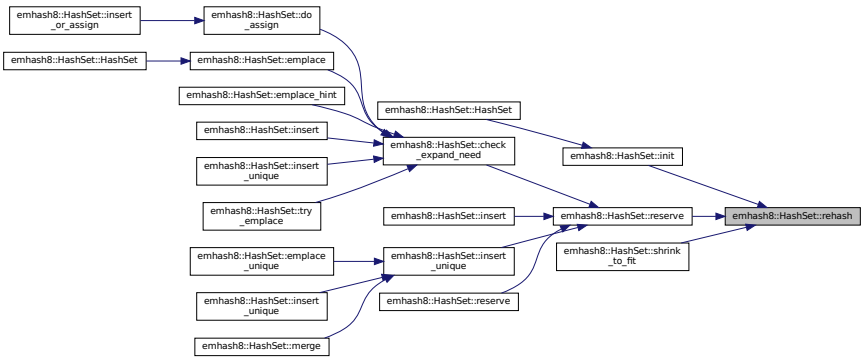
```
void emhash8::HashSet< KeyT, HashT, EqT >::rehash (
            uint64_t required_buckets )  [inline]
```

Definition at line 893 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

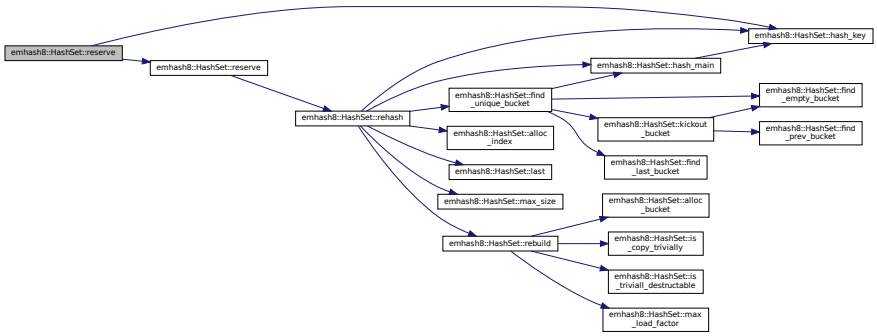


6.4.4.78 reserve() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
bool emhash8::HashSet< KeyT, HashT, EqT >::reserve (
            size_type required_buckets )  [inline]
```

Definition at line 843 of file emhash_set8.h.

Here is the call graph for this function:



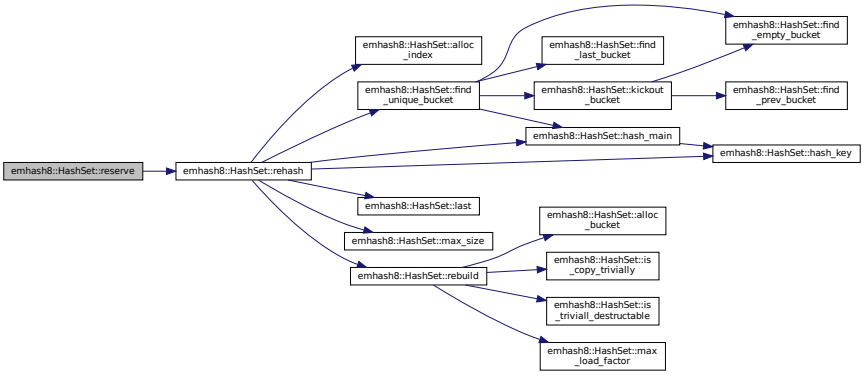
6.4.4.79 reserve() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
bool emhash8::HashSet< KeyT, HashT, EqT >::reserve (
            uint64_t num_elems,
            bool force )  [inline]
```

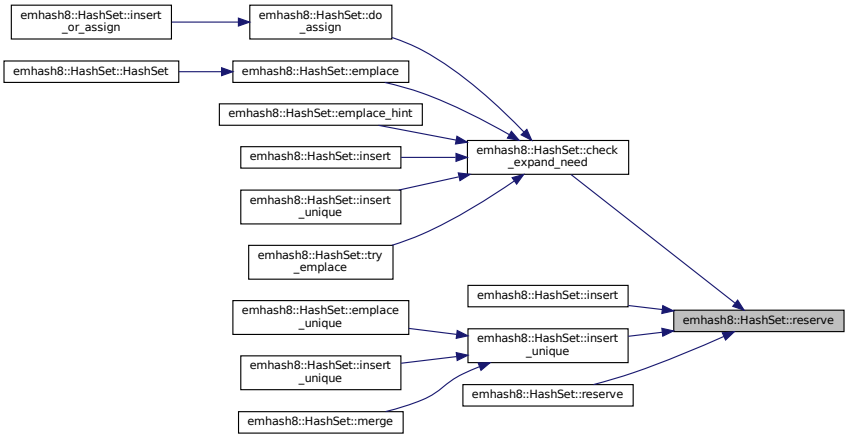
Make room for this many elements.

Definition at line 811 of file emhash_set8.h.

Here is the call graph for this function:



Here is the caller graph for this function:

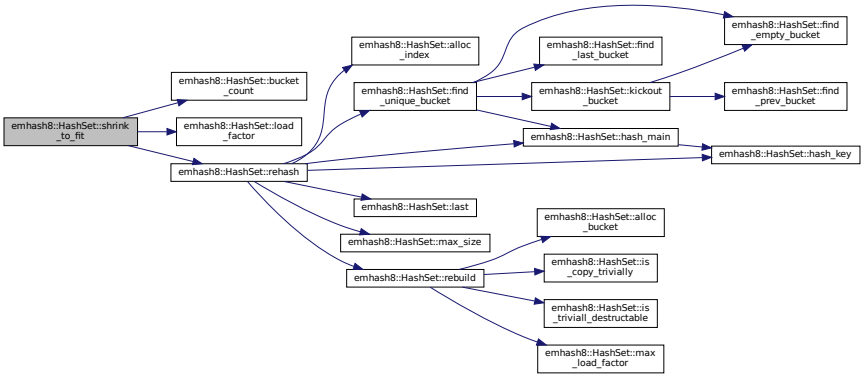


6.4.4.80 shrink_to_fit()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
void emhash8::HashSet< KeyT, HashT, EqT >::shrink_to_fit (
            const float min_factor = EMH_DEFAULT_LOAD_FACTOR / 4 )  [inline]
```

Definition at line 804 of file emhash_set8.h.

Here is the call graph for this function:

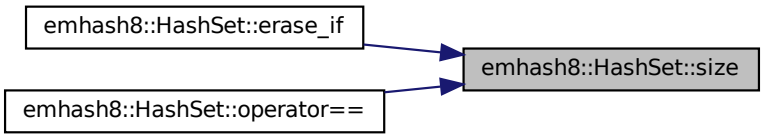


6.4.4.81 size()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::size ( ) const  [inline]
```

Definition at line 359 of file emhash_set8.h.

Here is the caller graph for this function:



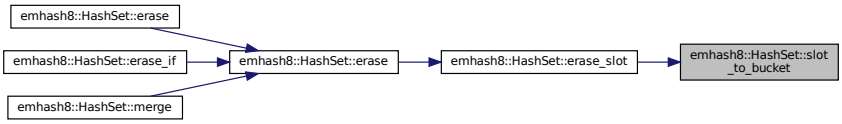
6.4.4.82 slot_to_bucket()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::slot_to_bucket (
            const size_type slot ) const  [inline], [private]
```

Definition at line 964 of file emhash_set8.h.

Here is the call graph for this function:

Here is the caller graph for this function:

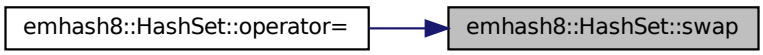


6.4.4.83 swap()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
void emhash8::HashSet< KeyT, HashT, EqT >::swap (
            HashSet< KeyT, HashT, EqT > & rhs )  [inline]
```

Definition at line 334 of file emhash_set8.h.

Here is the caller graph for this function:

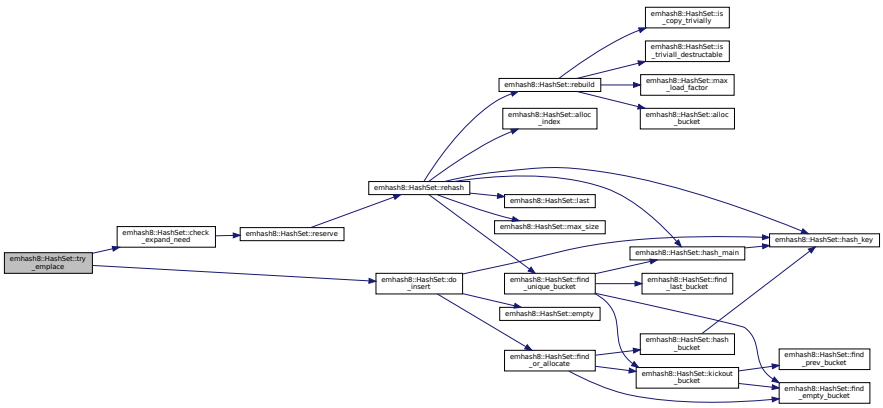


6.4.4.84 try_emplace() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
template<class...  Args>
std::pair<iterator, bool> emhash8::HashSet< KeyT, HashT, EqT >::try_emplace (
            const KeyT & k,
            Args &&...  args )  [inline]
```

Definition at line 688 of file emhash_set8.h.

Here is the call graph for this function:

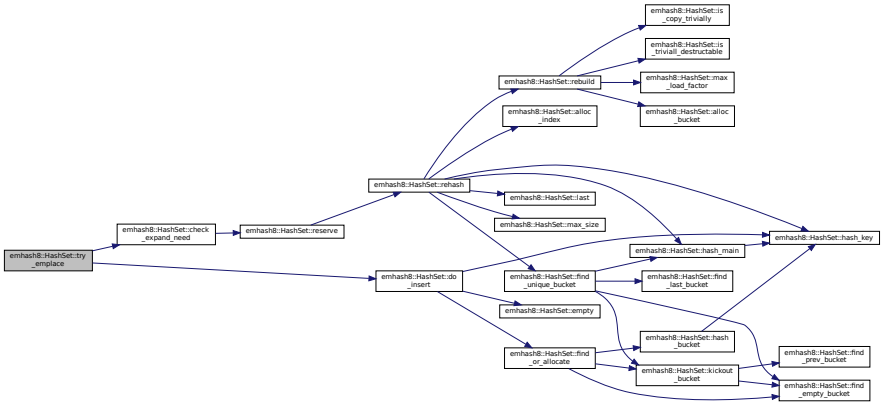


6.4.4.85 try_emplace() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
template<class... Args>
std::pair<iterator, bool> emhash8::HashSet< KeyT, HashT, EqT >::try_emplace (
            KeyT && k,
            Args &&... args ) [inline]
```

Definition at line 695 of file emhash_set8.h.

Here is the call graph for this function:



6.4.5 Member Data Documentation

6.4.5.1 _eq

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
EqT emhash8::HashSet< KeyT, HashT, EqT >::_eq [private]
```

Definition at line 1517 of file emhash_set8.h.

6.4.5.2 _hasher

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
HashT emhash8::HashSet< KeyT, HashT, EqT >::_hasher [private]
```

Definition at line 1516 of file emhash_set8.h.

6.4.5.3 _index

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
Index* emhash8::HashSet< KeyT, HashT, EqT >::_index  [private]
```

Definition at line 1514 of file emhash_set8.h.

6.4.5.4 _last

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::_last  [private]
```

Definition at line 1522 of file emhash_set8.h.

6.4.5.5 _mask

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::_mask  [private]
```

Definition at line 1519 of file emhash_set8.h.

6.4.5.6 _mlf

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
uint32_t emhash8::HashSet< KeyT, HashT, EqT >::_mlf  [private]
```

Definition at line 1518 of file emhash_set8.h.

6.4.5.7 _num_buckets

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::_num_buckets  [private]
```

Definition at line 1520 of file emhash_set8.h.

6.4.5.8 _num_filled

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::_num_filled  [private]
```

Definition at line 1521 of file emhash_set8.h.

6.4.5.9 _pairs

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
value_type* emhash8::HashSet< KeyT, HashT, EqT >::_pairs  [private]
```

Definition at line 1513 of file emhash_set8.h.

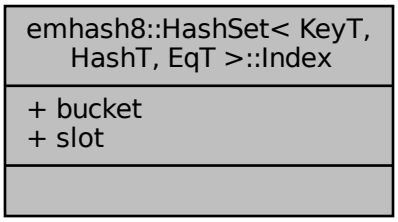
The documentation for this class was generated from the following file:

- include/emhash_set8.h

6.5 emhash8::HashSet< KeyT, HashT, EqT >::Index Struct Reference

```
#include <emhash_set8.h>
```

Collaboration diagram for emhash8::HashSet< KeyT, HashT, EqT >::Index:



Public Attributes

- size_type bucket
- size_type slot

6.5.1 Detailed Description

template<typename KeyT, typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
struct emhash8::HashSet< KeyT, HashT, EqT >::Index

Definition at line 103 of file emhash_set8.h.

6.5.2 Member Data Documentation

6.5.2.1 bucket

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::Index::bucket
```

Definition at line 105 of file emhash_set8.h.

6.5.2.2 slot

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
size_type emhash8::HashSet< KeyT, HashT, EqT >::Index::slot
```

Definition at line 106 of file emhash_set8.h.

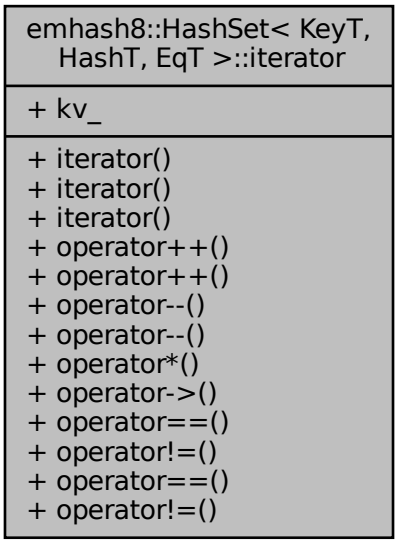
The documentation for this struct was generated from the following file:

- include/emhash_set8.h

6.6 emhash8::HashSet< KeyT, HashT, EqT >::iterator Class Reference

```
#include <emhash_set8.h>
```

Collaboration diagram for emhash8::HashSet< KeyT, HashT, EqT >::iterator:



Public Types

- using iterator_category = std::bidirectional_iterator_tag
- using difference_type = std::ptrdiff_t
- using value_type = typename htype::value_type
- using pointer = value_type *
- using const_pointer = const value_type *
- using reference = value_type &
- using const_reference = const value_type &

Public Member Functions

- iterator ()
- iterator (const_iterator &cit)
- iterator (const htype *hash_map, size_type bucket)
- iterator & operator++ ()
- iterator operator++ (int)
- iterator & operator-- ()
- iterator operator-- (int)
- reference operator* () const
- pointer operator-> () const
- bool operator== (const iterator &rhs) const
- bool operator!= (const iterator &rhs) const
- bool operator== (const const_iterator &rhs) const
- bool operator!= (const const_iterator &rhs) const

Public Attributes

- value_type * kv_

6.6.1 Detailed Description

template<typename KeyT, typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
class emhash8::HashSet< KeyT, HashT, EqT >::iterator

Definition at line 110 of file emhash_set8.h.

6.6.2 Member Typedef Documentation

6.6.2.1 const_pointer

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::iterator::const_pointer = const value_type*
```

Definition at line 117 of file emhash_set8.h.

6.6.2.2 const_reference

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::iterator::const_reference = const value_type&
```

Definition at line 119 of file emhash_set8.h.

6.6.2.3 difference_type

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::iterator::difference_type = std::ptrdiff_t
```

Definition at line 114 of file emhash_set8.h.

6.6.2.4 iterator_category

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::iterator::iterator_category = std::bidirectional↩
_iterator_tag
```

Definition at line 113 of file emhash_set8.h.

6.6.2.5 pointer

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::iterator::pointer = value_type*
```

Definition at line 116 of file emhash_set8.h.

6.6.2.6 reference

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::iterator::reference = value_type&
```

Definition at line 118 of file emhash_set8.h.

6.6.2.7 value_type

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
using emhash8::HashSet< KeyT, HashT, EqT >::iterator::value_type = typename htype::value_type
```

Definition at line 115 of file emhash_set8.h.

6.6.3 Constructor & Destructor Documentation

6.6.3.1 iterator() [1/3]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
emhash8::HashSet< KeyT, HashT, EqT >::iterator::iterator ( ) [inline]
```

Definition at line 121 of file emhash_set8.h.

6.6.3.2 iterator() [2/3]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
emhash8::HashSet< KeyT, HashT, EqT >::iterator::iterator (
            const_iterator & cit )  [inline]
```

Definition at line 122 of file emhash_set8.h.

6.6.3.3 iterator() [3/3]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
emhash8::HashSet< KeyT, HashT, EqT >::iterator::iterator (
            const htype * hash_map,
            size_type bucket )  [inline]
```

Definition at line 126 of file emhash_set8.h.

6.6.4 Member Function Documentation

6.6.4.1 operator"!=() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
bool emhash8::HashSet< KeyT, HashT, EqT >::iterator::operator!= (
            const const_iterator & rhs ) const  [inline]
```

Definition at line 160 of file emhash_set8.h.

6.6.4.2 operator"!=() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<KeyT>>
bool emhash8::HashSet< KeyT, HashT, EqT >::iterator::operator!= (
            const iterator & rhs ) const  [inline]
```

Definition at line 158 of file emhash_set8.h.

6.6.4.3 operator*()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
reference emhash8::HashSet< KeyT, HashT, EqT >::iterator::operator* ( ) const  [inline]
```

Definition at line 154 of file emhash_set8.h.

6.6.4.4 operator++() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
iterator& emhash8::HashSet< KeyT, HashT, EqT >::iterator::operator++ ( )  [inline]
```

Definition at line 130 of file emhash_set8.h.

6.6.4.5 operator++() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
iterator emhash8::HashSet< KeyT, HashT, EqT >::iterator::operator++ (
            int  )  [inline]
```

Definition at line 136 of file emhash_set8.h.

6.6.4.6 operator--() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
iterator& emhash8::HashSet< KeyT, HashT, EqT >::iterator::operator-- ( )  [inline]
```

Definition at line 142 of file emhash_set8.h.

6.6.4.7 operator--() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
iterator emhash8::HashSet< KeyT, HashT, EqT >::iterator::operator-- (
            int  )  [inline]
```

Definition at line 148 of file emhash_set8.h.

6.6.4.8 operator->()

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
pointer emhash8::HashSet< KeyT, HashT, EqT >::iterator::operator-> ( ) const  [inline]
```

Definition at line 155 of file emhash_set8.h.

6.6.4.9 operator==() [1/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
bool emhash8::HashSet< KeyT, HashT, EqT >::iterator::operator== (
            const const_iterator & rhs ) const  [inline]
```

Definition at line 159 of file emhash_set8.h.

6.6.4.10 operator==() [2/2]

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
bool emhash8::HashSet< KeyT, HashT, EqT >::iterator::operator== (
            const iterator & rhs ) const  [inline]
```

Definition at line 157 of file emhash_set8.h.

6.6.5 Member Data Documentation

6.6.5.1 kv_

```
template<typename KeyT , typename HashT = std::hash<KeyT>, typename EqT = std::equal_to<Key↩
T>>
value_type* emhash8::HashSet< KeyT, HashT, EqT >::iterator::kv_
```

Definition at line 163 of file emhash_set8.h.

The documentation for this class was generated from the following file:

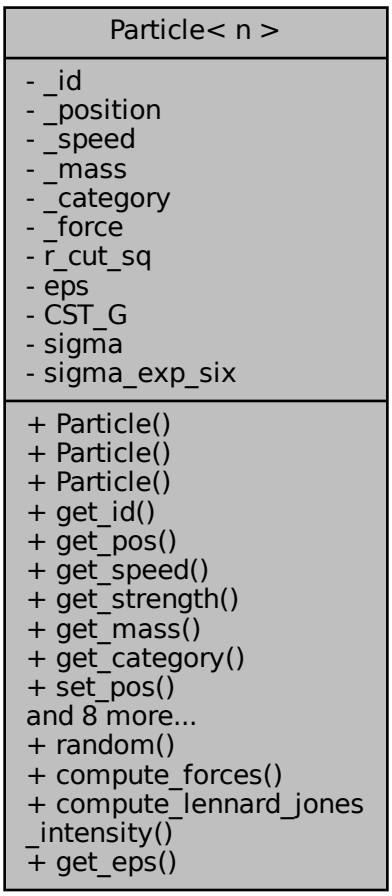
- include/emhash_set8.h

6.7 Particle< n > Class Template Reference

Represents a particle in an n-dimensional world, described by its position, speed, mass, and category.

`#include <Particle.h>`

Collaboration diagram for Particle< n >:



Public Member Functions

- Particle (uint32_t id, Vector< n > pos, Vector< n > speed, double mass, Category category)
- Particle ()=default

  Constructs a default Particle object.
- Particle (const Particle &p)=default

  Constructs a Particle object by copying another Particle object.
- uint32_t get_id ()

  Gets the ID of the particle.
- const Vector< n > & get_pos ()

Gets the position of the particle in the n-dimensional world.

- const Vector< n > & get_speed ()

  Gets the speed of the particle in the n-dimensional world.
- const Vector< n > & get_strength ()

  Gets the force acting on the particle in the n-dimensional world.
- const double & get_mass ()

  Gets the mass of the particle.
- const Category & get_category ()

  Gets the category (type) of the particle.
- void set_pos (const Vector< n > &new_pos)

  Sets the position of the particle in the n-dimensional world.
- void set_speed (const Vector< n > &new_speed)

  Sets the speed of the particle in the n-dimensional world.
- void set_force (const Vector< n > &new_force)
- void apply_force (const Vector< n > &new_force)
- bool operator< (const Particle &p) const
- bool operator== (const Particle &p) const
- bool operator!= (const Particle &p) const
- double kinetic_energy ()
- void apply_gravity ()

Static Public Member Functions

- static Particle random (uint32_t id)

  Generates a random Particle object with the given ID.
- static void compute_forces (Particle &a, Particle &b, bool gravitational, bool lennard_jones)

  Compute the force that each particule exert on the other and add it to the particle strength field.
- static double compute_lennard_jones_intensity (double dist_sq)
- static double get_eps ()

Private Attributes

- uint32_t _id {}
- Vector< n > _position
- Vector< n > _speed
- double _mass {}
- Category _category
- Vector< n > _force

Static Private Attributes

- static double r_cut_sq = 2.5∗2.5
- static double eps = 5.0
- static double CST_G = -9.81
- static double sigma = 1.0
- static double sigma_exp_six = pow(sigma, 6)

Friends

- std::ostream & operator<< (std::ostream &os, const Particle &p)

6.7.1 Detailed Description

template<unsigned int n>
class Particle< n >

Represents a particle in an n-dimensional world, described by its position, speed, mass, and category.

Template Parameters

n	The dimension of the world in which the particle exists.

Definition at line 33 of file Particle.h.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 Particle() [1/3]

```
template<unsigned int n>
Particle< n >::Particle (
            uint32_t id,
            Vector< n > pos,
            Vector< n > speed,
            double mass,
            Category category )
```

6.7.2.2 Particle() [2/3]

```
template<unsigned int n>
Particle< n >::Particle ( )  [default]
```

Constructs a default Particle object.

6.7.2.3 Particle() [3/3]

```
template<unsigned int n>
Particle< n >::Particle (
            const Particle< n > & p )  [default]
```

Constructs a Particle object by copying another Particle object.

Parameters

p	The Particle object to copy.

6.7.3 Member Function Documentation

6.7.3.1 apply_force()

```
template<unsigned int n>
void Particle< n >::apply_force (
            const Vector< n > & new_force )  [inline]
```

Definition at line 131 of file Particle.h.

6.7.3.2 apply_gravity()

```
template<unsigned int n>
void Particle< n >::apply_gravity ( )  [inline]
```

6.7.3.3 compute_forces()

```
template<unsigned int n>
static void Particle< n >::compute_forces (
            Particle< n > & a,
            Particle< n > & b,
            bool gravitational,
            bool lennard_jones )  [static]
```

Compute the force that each particule exert on the other and add it to the particle strength field.

Parameters

a	Particule a
b	Particule a
gravitational	should the gravitationnal force be taken into account
lennard_jones	should the lennard_jones potential related force be taken into account

6.7.3.4 compute_lennard_jones_intensity()

```
template<unsigned int n>
static double Particle< n >::compute_lennard_jones_intensity (
            double dist_sq )  [inline], [static]
```

6.7.3.5 get_category()

```
template<unsigned int n>
const Category& Particle< n >::get_category ( )  [inline]
```

Gets the category (type) of the particle.

Returns

The category (type) of the particle.

Definition at line 107 of file Particle.h.

6.7.3.6 get_eps()

```
template<unsigned int n>
static double Particle< n >::get_eps ( )  [inline], [static]
```

Definition at line 173 of file Particle.h.

6.7.3.7 get_id()

```
template<unsigned int n>
uint32_t Particle< n >::get_id ( )  [inline]
```

Gets the ID of the particle.

Returns

The ID of the particle.

Definition at line 67 of file Particle.h.

6.7.3.8 get_mass()

```
template<unsigned int n>
const double& Particle< n >::get_mass ( )  [inline]
```

Gets the mass of the particle.

Returns

The mass of the particle.

Definition at line 99 of file Particle.h.

6.7.3.9 get_pos()

```
template<unsigned int n>
const Vector<n>& Particle< n >::get_pos ( )  [inline]
```

Gets the position of the particle in the n-dimensional world.

Returns

The position of the particle.

Definition at line 75 of file Particle.h.

6.7.3.10 get_speed()

```
template<unsigned int n>
const Vector<n>& Particle< n >::get_speed ( )  [inline]
```

Gets the speed of the particle in the n-dimensional world.

Returns

The speed of the particle.

Definition at line 83 of file Particle.h.

6.7.3.11 get_strength()

```
template<unsigned int n>
const Vector<n>& Particle< n >::get_strength ( )  [inline]
```

Gets the force acting on the particle in the n-dimensional world.

Returns

The force acting on the particle.

Definition at line 91 of file Particle.h.

6.7.3.12 kinetic_energy()

```
template<unsigned int n>
double Particle< n >::kinetic_energy ( )  [inline]
```

6.7.3.13 operator"!=()

```
template<unsigned int n>
bool Particle< n >::operator!= (
            const Particle< n > & p ) const  [inline]
```

Definition at line 143 of file Particle.h.

6.7.3.14 operator<()

```
template<unsigned int n>
bool Particle< n >::operator< (
            const Particle< n > & p ) const  [inline]
```

Definition at line 135 of file Particle.h.

6.7.3.15 operator==()

```
template<unsigned int n>
bool Particle< n >::operator== (
            const Particle< n > & p ) const  [inline]
```

Definition at line 139 of file Particle.h.

6.7.3.16 random()

```
template<unsigned int n>
static Particle Particle< n >::random (
            uint32_t id )  [static]
```

Generates a random Particle object with the given ID.

Parameters

id	The ID of the particle.

Returns

A new Particle object with random position, speed, mass, and category.

6.7.3.17 set_force()

```
template<unsigned int n>
void Particle< n >::set_force (
            const Vector< n > & new_force )  [inline]
```

Definition at line 127 of file Particle.h.

6.7.3.18 set_pos()

```
template<unsigned int n>
void Particle< n >::set_pos (
            const Vector< n > & new_pos )  [inline]
```

Sets the position of the particle in the n-dimensional world.

Parameters

new_pos	The new position of the particle.

Definition at line 115 of file Particle.h.

6.7.3.19 set_speed()

```
template<unsigned int n>
void Particle< n >::set_speed (
            const Vector< n > & new_speed )  [inline]
```

Sets the speed of the particle in the n-dimensional world.

Parameters

new_speed	The new speed

Definition at line 123 of file Particle.h.

6.7.4 Friends And Related Function Documentation

6.7.4.1 operator<<

```
template<unsigned int n>
std::ostream& operator<< (
            std::ostream & os,
            const Particle< n > & p )  [friend]
```

Definition at line 147 of file Particle.h.

6.7.5 Member Data Documentation

6.7.5.1 _category

```
template<unsigned int n>
Category Particle< n >::_category  [private]
```

Definition at line 188 of file Particle.h.

6.7.5.2 _force

```
template<unsigned int n>
Vector<n> Particle< n >::_force  [private]
```

Definition at line 189 of file Particle.h.

6.7.5.3 _id

```
template<unsigned int n>
uint32_t Particle< n >::_id {}  [private]
```

Definition at line 184 of file Particle.h.

6.7.5.4 _mass

```
template<unsigned int n>
double Particle< n >::_mass {}  [private]
```

Definition at line 187 of file Particle.h.

6.7.5.5 _position

```
template<unsigned int n>
Vector<n> Particle< n >::_position  [private]
```

Definition at line 185 of file Particle.h.

6.7.5.6 _speed

```
template<unsigned int n>
Vector<n> Particle< n >::_speed  [private]
```

Definition at line 186 of file Particle.h.

6.7.5.7 CST_G

```
template<unsigned int n>
double Particle< n >::CST_G = -9.81  [inline], [static], [private]
```

Definition at line 180 of file Particle.h.

6.7.5.8 eps

```
template<unsigned int n>
double Particle< n >::eps = 5.0  [inline], [static], [private]
```

Definition at line 179 of file Particle.h.

6.7.5.9 r_cut_sq

```
template<unsigned int n>
double Particle< n >::r_cut_sq = 2.5*2.5  [inline], [static], [private]
```

Definition at line 176 of file Particle.h.

6.7.5.10 sigma

```
template<unsigned int n>
double Particle< n >::sigma = 1.0  [inline], [static], [private]
```

Definition at line 181 of file Particle.h.

6.7.5.11 sigma_exp_six

```
template<unsigned int n>
double Particle< n >::sigma_exp_six = pow(sigma, 6)  [inline], [static], [private]
```

Definition at line 182 of file Particle.h.

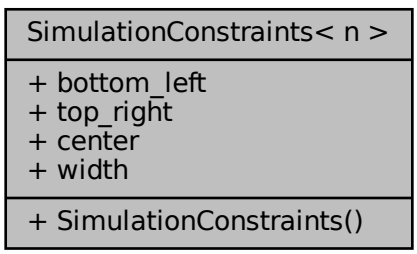
The documentation for this class was generated from the following file:

- include/Particle.h

6.8 SimulationConstraints< n > Struct Template Reference

```
#include <Universe.h>
```

Collaboration diagram for SimulationConstraints< n >:



Public Member Functions

- SimulationConstraints (Vector< n > bottom_left, Vector< n > top_right)

Public Attributes

- Vector< n > bottom_left

  Particle position can't go below this.
- Vector< n > top_right

  Particle position can't go above this.
- Vector< n > center

  The center of the universe.
- Vector< n > width

  The distance from the center to the universe's borders for each dimension.

6.8.1 Detailed Description

template<unsigned int n>
struct SimulationConstraints< n >

Definition at line 41 of file Universe.h.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 SimulationConstraints()

```
template<unsigned int n>
SimulationConstraints< n >::SimulationConstraints (
            Vector< n > bottom_left,
            Vector< n > top_right )  [inline]
```

Definition at line 43 of file Universe.h.

6.8.3 Member Data Documentation

6.8.3.1 bottom_left

```
template<unsigned int n>
Vector<n> SimulationConstraints< n >::bottom_left
```

Particle position can't go below this.

Definition at line 46 of file Universe.h.

6.8.3.2 center

```
template<unsigned int n>
Vector<n> SimulationConstraints< n >::center
```

The center of the universe.

Definition at line 48 of file Universe.h.

6.8.3.3 top_right

```
template<unsigned int n>
Vector<n> SimulationConstraints< n >::top_right
```

Particle position can't go above this.

Definition at line 47 of file Universe.h.

6.8.3.4 width

```
template<unsigned int n>
Vector<n> SimulationConstraints< n >::width
```

The distance from the center to the universe's borders for each dimension.

Definition at line 49 of file Universe.h.

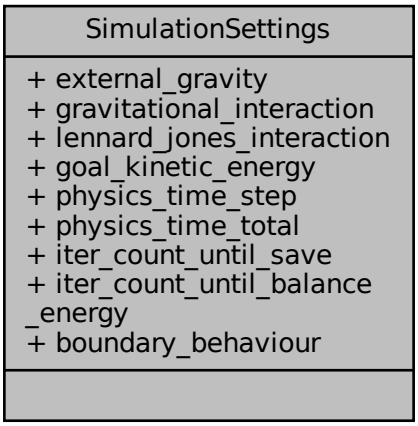
The documentation for this struct was generated from the following file:

- include/Universe.h

6.9 SimulationSettings Struct Reference

```
#include <Universe.h>
```

Collaboration diagram for SimulationSettings:



Public Attributes

- bool external_gravity = false

  Should the external gravity be applied.
- bool gravitational_interaction = false

  Should the gravitational interaction be applied.
- bool lennard_jones_interaction = false

  Should the Lennard-Jones interaction be applied.
- double goal_kinetic_energy = -1

  The kinetic energy goal set by the user to force the system's kinetic energy to this level every 1000 physics iterations.
- double physics_time_step = -1.

  The time step used in the physics simulation.
- double physics_time_total = -1.

  The total time for the physics simulation.
- uint32_t iter_count_until_save = 0

  The number of iterations until saving the simulation.
- uint32_t iter_count_until_balance_energy = 1000

  The number of iterations until balancing the system's kinetic energy.
- BoundaryBehaviour boundary_behaviour = Absorption

  Particle behavior if outside of boundaries.

6.9.1 Detailed Description

Definition at line 27 of file Universe.h.

6.9.2 Member Data Documentation

6.9.2.1 boundary_behaviour

```
BoundaryBehaviour SimulationSettings::boundary_behaviour = Absorption
```

Particle behavior if outside of boundaries.

Definition at line 37 of file Universe.h.

6.9.2.2 external_gravity

```
bool SimulationSettings::external_gravity = false
```

Should the external gravity be applied.

Definition at line 29 of file Universe.h.

6.9.2.3 goal_kinetic_energy

```
double SimulationSettings::goal_kinetic_energy = -1
```

The kinetic energy goal set by the user to force the system's kinetic energy to this level every 1000 physics iterations.

Definition at line 32 of file Universe.h.

6.9.2.4 gravitational_interaction

```
bool SimulationSettings::gravitational_interaction = false
```

Should the gravitational interaction be applied.

Definition at line 30 of file Universe.h.

6.9.2.5 iter_count_until_balance_energy

```
uint32_t SimulationSettings::iter_count_until_balance_energy = 1000
```

The number of iterations until balancing the system's kinetic energy.

Definition at line 36 of file Universe.h.

6.9.2.6 iter_count_until_save

```
uint32_t SimulationSettings::iter_count_until_save = 0
```

The number of iterations until saving the simulation.

Definition at line 35 of file Universe.h.

6.9.2.7 lennard_jones_interaction

```
bool SimulationSettings::lennard_jones_interaction = false
```

Should the Lennard-Jones interaction be applied.

Definition at line 31 of file Universe.h.

6.9.2.8 physics_time_step

```
double SimulationSettings::physics_time_step = -1.
```

The time step used in the physics simulation.

Definition at line 33 of file Universe.h.

6.9.2.9 physics_time_total

```
double SimulationSettings::physics_time_total = -1.
```

The total time for the physics simulation.

Definition at line 34 of file Universe.h.

The documentation for this struct was generated from the following file:

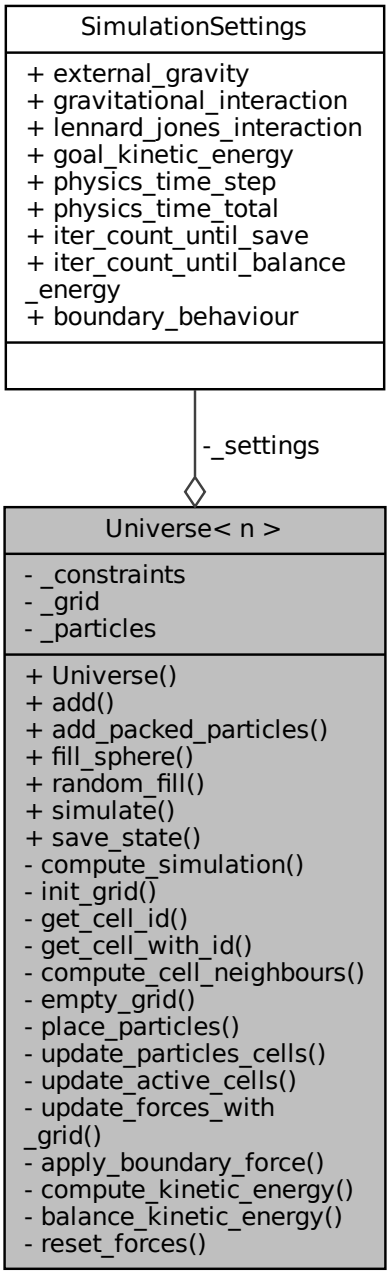
- include/Universe.h

6.10 Universe< n > Class Template Reference

Class representing the simulation universe.

```
#include <Universe.h>
```

Collaboration diagram for Universe< n >:



Public Member Functions

- Universe (SimulationConstraints< n > constraints, double cell_size)
- void add (Vector< n > position, Vector< n > velocity, double mass, Category category)
- int add_packed_particles (Vector< n > bottom_left, Vector< n > top_right, Vector< n > velocity, double mass, Category category, Vector< n > particle_count)

- int fill_sphere (Vector< n > bottom_left, Vector< n > top_right, Vector< n > velocity, double mass, Category category, Vector< n > particle_count)
- void random_fill (uint32_t particle_count)
- void simulate (SimulationSettings settings)
- void save_state (const std::string &filename)

Private Member Functions

- void compute_simulation ()
- void init_grid (double cell_size)

  Initialize the universe grid with the dimensions `size`
- int32_t get_cell_id (Vector< n > position)

  Compute the id of the cell that contains `position`
- Cell get_cell_with_id (int32_t id)

  Getter for the cell with id `id`
- void compute_cell_neighbours (int32_t id, Cell &cell)

  Compute the neightbour cell of cell with id `id`
- void empty_grid ()

  Empty the grid.
- void place_particles ()

  place all the universe's particles in their coresponding cell
- void update_particles_cells ()

  move all the universe's particles in their coresponding cell
- void update_active_cells ()
- void update_forces_with_grid ()

  Update all the particles forces field using the grid.
- void apply_boundary_force (Particle< n > &particle, const std::vector< bool > &boundaries)
- double compute_kinetic_energy ()
- void balance_kinetic_energy ()
- void reset_forces ()

  Reset all particles forces to zero.

Private Attributes

- SimulationSettings _settings

  The simulation settings.
- SimulationConstraints< n > _constraints

  The simulation constraints.
- Grid< n > _grid

  The grid characteristics.
- std::vector< Particle< n > > _particles

  The universe particles.

6.10.1 Detailed Description

template<unsigned int n>
class Universe< n >

Class representing the simulation universe.

Definition at line 58 of file Universe.h.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 Universe()

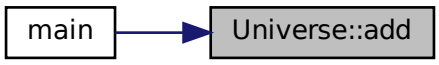
```
template<unsigned int n>
Universe< n >::Universe (
            SimulationConstraints< n > constraints,
            double cell_size )
```

6.10.3 Member Function Documentation

6.10.3.1 add()

```
template<unsigned int n>
void Universe< n >::add (
            Vector< n > position,
            Vector< n > velocity,
            double mass,
            Category category )
```

Here is the caller graph for this function:



6.10.3.2 add_packed_particles()

```
template<unsigned int n>
int Universe< n >::add_packed_particles (
            Vector< n > bottom_left,
            Vector< n > top_right,
            Vector< n > velocity,
            double mass,
            Category category,
            Vector< n > particle_count )
```

Here is the caller graph for this function:

6.10.3.3 apply_boundary_force()

```
template<unsigned int n>
void Universe< n >::apply_boundary_force (
            Particle< n > & particle,
            const std::vector< bool > & boundaries )  [private]
```

6.10.3.4 balance_kinetic_energy()

```
template<unsigned int n>
void Universe< n >::balance_kinetic_energy ( )  [private]
```

6.10.3.5 compute_cell_neighbours()

```
template<unsigned int n>
void Universe< n >::compute_cell_neighbours (
            int32_t id,
            Cell & cell )  [private]
```

Compute the neightbour cell of cell with id `id`

Parameters

id	

6.10.3.6 compute_kinetic_energy()

```
template<unsigned int n>
double Universe< n >::compute_kinetic_energy ( )  [private]
```

6.10.3.7 compute_simulation()

```
template<unsigned int n>
void Universe< n >::compute_simulation ( )  [private]
```

6.10.3.8 empty_grid()

```
template<unsigned int n>
void Universe< n >::empty_grid ( )  [private]
```

Empty the grid.

6.10.3.9 fill_sphere()

```
template<unsigned int n>
int Universe< n >::fill_sphere (
            Vector< n > bottom_left,
            Vector< n > top_right,
            Vector< n > velocity,
            double mass,
            Category category,
            Vector< n > particle_count )
```

6.10.3.10 get_cell_id()

```
template<unsigned int n>
int32_t Universe< n >::get_cell_id (
            Vector< n > position )  [private]
```

Compute the id of the cell that contains `position`

Parameters

postion	

6.10.3.11 get_cell_with_id()

```
template<unsigned int n>
```

```
Cell Universe< n >::get_cell_with_id (
            int32_t id )  [private]
```

Getter for the cell with id `id`

Parameters

id	

6.10.3.12 init_grid()

```
template<unsigned int n>
void Universe< n >::init_grid (
            double cell_size )  [private]
```

Initialize the universe grid with the dimensions `size`

Parameters

size	the dimensions of the grid

6.10.3.13 place_particles()

```
template<unsigned int n>
void Universe< n >::place_particles ( )  [private]
```

place all the universe's particles in their coresponding cell

6.10.3.14 random_fill()

```
template<unsigned int n>
void Universe< n >::random_fill (
            uint32_t particle_count )
```

Here is the caller graph for this function:

6.10.3.15 reset_forces()

```
template<unsigned int n>
void Universe< n >::reset_forces ( )  [private]
```

Reset all particles forces to zero.

6.10.3.16 save_state()

```
template<unsigned int n>
void Universe< n >::save_state (
            const std::string & filename )
```

6.10.3.17 simulate()

```
template<unsigned int n>
void Universe< n >::simulate (
            SimulationSettings settings )
```

Here is the caller graph for this function:

6.10.3.18 update_active_cells()

```
template<unsigned int n>
void Universe< n >::update_active_cells ( )  [private]
```

6.10.3.19 update_forces_with_grid()

```
template<unsigned int n>
void Universe< n >::update_forces_with_grid ( )  [private]
```

Update all the particles forces field using the grid.

Parameters

gravitational	should the gravitational force be taken into account
lennard_jones	should the lennard_jones potential force be taken into account

6.10.3.20 update_particles_cells()

```
template<unsigned int n>
void Universe< n >::update_particles_cells ( )  [private]
```

move all the universe's particles in their coresponding cell

6.10.4 Member Data Documentation

6.10.4.1 _constraints

```
template<unsigned int n>
SimulationConstraints<n> Universe< n >::_constraints  [private]
```

The simulation constraints.

Definition at line 76 of file Universe.h.

6.10.4.2 _grid

```
template<unsigned int n>
Grid<n> Universe< n >::_grid  [private]
```

The grid characteristics.

Definition at line 77 of file Universe.h.

6.10.4.3 _particles

```
template<unsigned int n>
std::vector<Particle<n> > Universe< n >::_particles  [private]
```

The universe particles.

Definition at line 79 of file Universe.h.

6.10.4.4 _settings

```
template<unsigned int n>
SimulationSettings Universe< n >::_settings  [private]
```

The simulation settings.

Definition at line 75 of file Universe.h.

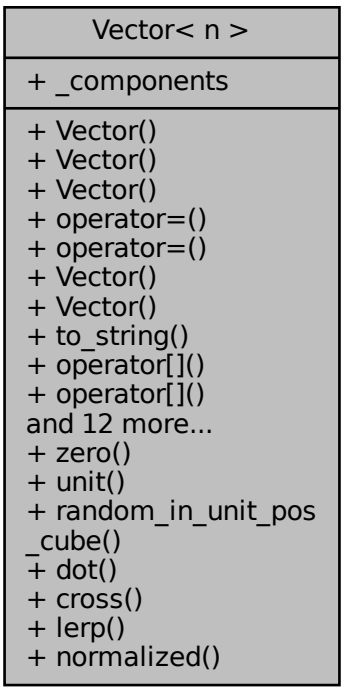
The documentation for this class was generated from the following file:

- include/Universe.h

6.11 Vector< n > Class Template Reference

```
#include <Vector.h>
```

Collaboration diagram for Vector< n >:



Public Member Functions

- Vector ()=default

  Default constructor.
- Vector (double a)

  Constructor that initializes all components with the same value.
- Vector (Vector< n > &&other) noexcept

  Move constructor for Vector class.
- Vector< n > & operator= (Vector< n > &&other) noexcept

  Move assignment operator for Vector class.
- Vector< n > & operator= (const Vector< n > &other)

  Copy assignment operator for Vector class.
- template<typename... Args>
  Vector (Args... args)

  Variadic constructor that initializes the components with the given values.
- Vector (const Vector &v)

  Copy constructor.
- std::string to_string () const

  Returns a string representation of the vector.
- double operator[ ] (uint32_t i) const

  Returns the value of the i-th component of the vector.
- double & operator[ ] (uint32_t i)

  Returns a reference to the i-th component of the vector.
- Vector & operator+= (const Vector &other)

  Adds another vector to this vector.
- Vector & operator-= (const Vector &other)

  Subtracts another vector from this vector.
- Vector operator- () const

  Negates this vector.
- Vector & operator*= (const Vector &other)

  Multiplies this vector with another vector component-wise.
- Vector & operator*= (double scalar)

  Multiplies this vector with a scalar.
- Vector & operator/= (double scalar)

  Divides this vector by a scalar.
- Vector & operator/= (Vector< n > v)

  Divides each component of this vector by each component of v.
- Vector & operator%= (const Vector< n > &v)

  Apply modulo v[i] to each component i of this vector.
- Vector & operator%= (double v)

  Apply modulo v to each component of this vector.
- double sq_length () const

  sq_length Returns the square of the length of the vector.
- double length () const

  length Returns the length of the vector.
- Vector & normalize ()

  normalize Normalizes the vector.

Static Public Member Functions

- static Vector zero ()

  Returns a zero vector.
- static Vector unit (uint32_t i)

  Returns a vector with 1 on the i-dimension and 0 for all the other components.
- static Vector random_in_unit_pos_cube ()

  Returns a random vector in the unit positive cube.
- static double dot (const Vector &v1, const Vector &v2)

  dot Computes the dot product of two vectors.
- static Vector cross (const Vector &v1, const Vector &v2)

  cross Computes the cross product of two vectors.
- static Vector lerp (const Vector &v1, const Vector &v2, double x)

  lerp Performs a linear interpolation between two vectors.
- static Vector normalized (const Vector &v)

  normalized Returns a normalized copy of the vector.

Public Attributes

- double _components [n]

  Components of the vector.

Friends

- std::ostream & operator<< (std::ostream &out, const Vector &v)

  Overloaded output stream operator for printing the vector.
- bool operator== (const Vector &a, const Vector &b)

  operator== Compares two vectors for equality.
- bool operator!= (const Vector &a, const Vector &b)

  operator!= Compares two vectors for inequality.

6.11.1 Detailed Description

template<unsigned int n>
class Vector< n >

Definition at line 7 of file Vector.h.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Vector() [1/5]

```
template<unsigned int n>
Vector< n >::Vector ( )  [default]
```

Default constructor.

6.11.2.2 Vector() [2/5]

```
template<unsigned int n>
Vector< n >::Vector (
            double a )
```

Constructor that initializes all components with the same value.

Parameters

a	The value to initialize the components with.

6.11.2.3 Vector() [3/5]

```
template<unsigned int n>
Vector< n >::Vector (
            Vector< n > && other )  [noexcept]
```

Move constructor for Vector class.

Parameters

other	The other Vector object to move from.

6.11.2.4 Vector() [4/5]

```
template<unsigned int n>
template<typename... Args>
Vector< n >::Vector (
            Args... args )
```

Variadic constructor that initializes the components with the given values.

Template Parameters

Args	The types of the arguments.

Parameters

args	The values to initialize the components with.

6.11.2.5 Vector() [5/5]

```
template<unsigned int n>
Vector< n >::Vector (
            const Vector< n > & v )
```

Copy constructor.

Parameters

other	The vector to copy from.

6.11.3 Member Function Documentation

6.11.3.1 cross()

```
template<unsigned int n>
static Vector Vector< n >::cross (
            const Vector< n > & v1,
            const Vector< n > & v2 )  [static]
```

cross Computes the cross product of two vectors.

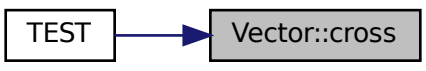
Parameters

v1	The first vector.
v2	The second vector.

Returns

The cross product of the two vectors.

Here is the caller graph for this function:



6.11.3.2 dot()

```
template<unsigned int n>
static double Vector< n >::dot (
             const Vector< n > & v1,
             const Vector< n > & v2 )  [static]
```

dot Computes the dot product of two vectors.

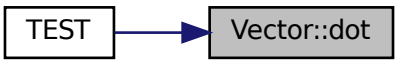
Parameters

v1	The first vector.
v2	The second vector.

Returns

The dot product of the two vectors.

Here is the caller graph for this function:



6.11.3.3 length()

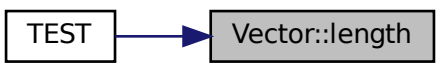
```
template<unsigned int n>
double Vector< n >::length ( ) const
```

length Returns the length of the vector.

Returns

The length of the vector.

Here is the caller graph for this function:



6.11.3.4 lerp()

```
template<unsigned int n>
static Vector Vector< n >::lerp (
            const Vector< n > & v1,
            const Vector< n > & v2,
            double x )  [static]
```

lerp Performs a linear interpolation between two vectors.

Parameters

v1	The first vector.
v2	The second vector.
x	The interpolation factor (should be between 0 and 1).

Returns

The interpolated vector.

6.11.3.5 normalize()

```
template<unsigned int n>
Vector& Vector< n >::normalize ( )
```

normalize Normalizes the vector.

Returns

A reference to the normalized vector.

Here is the caller graph for this function:

6.11.3.6 normalized()

```
template<unsigned int n>
static Vector Vector< n >::normalized (
            const Vector< n > & v )  [static]
```

normalized Returns a normalized copy of the vector.

Parameters

v	The vector to normalize.

Returns

The normalized copy of the vector.

Here is the caller graph for this function:

6.11.3.7 operator%=() [1/2]

```
template<unsigned int n>
Vector& Vector< n >::operator%= (
            const Vector< n > & v )
```

Apply modulo v[i] to each component i of this vector.

Parameters

scalar	v the divisor

Returns

A reference to this vector.

6.11.3.8 operator%=() [2/2]

```
template<unsigned int n>
Vector& Vector< n >::operator%= (
            double v )
```

Apply modulo v to each component of this vector.

Parameters

scalar	v

Returns

A reference to this vector.

6.11.3.9 operator*=() [1/2]

```
template<unsigned int n>
Vector& Vector< n >::operator*= (
            const Vector< n > & other )
```

Multiplies this vector with another vector component-wise.

Parameters

other	The vector to multiply with.

Returns

A reference to this vector.

6.11.3.10 operator*=() [2/2]

```
template<unsigned int n>
Vector& Vector< n >::operator*= (
            double scalar )
```

Multiplies this vector with a scalar.

Parameters

scalar	The scalar to multiply with.

Returns

A reference to this vector.

6.11.3.11 operator+=()

```
template<unsigned int n>
Vector& Vector< n >::operator+= (
            const Vector< n > & other )
```

Adds another vector to this vector.

Parameters

other	The vector to add.

Returns

A reference to this vector.

6.11.3.12 operator-()

```
template<unsigned int n>
Vector Vector< n >::operator- ( ) const
```

Negates this vector.

Returns

The negated vector.

6.11.3.13 operator-=()

```
template<unsigned int n>
Vector& Vector< n >::operator-= (
            const Vector< n > & other )
```

Subtracts another vector from this vector.

Parameters

other	The vector to subtract.

Returns

A reference to this vector.

6.11.3.14 operator/=() [1/2]

```
template<unsigned int n>
Vector& Vector< n >::operator/= (
            double scalar )
```

Divides this vector by a scalar.

Parameters

scalar	The scalar to divide by.

Returns

A reference to this vector.

6.11.3.15 operator/=() [2/2]

```
template<unsigned int n>
Vector& Vector< n >::operator/= (
            Vector< n > v )
```

Divides each component of this vector by each component of v.

Parameters

scalar	v the divisor

Returns

A reference to this vector.

6.11.3.16 operator=() [1/2]

```
template<unsigned int n>
Vector<n>& Vector< n >::operator= (
            const Vector< n > & other )
```

Copy assignment operator for Vector class.

Parameters

other	The other Vector object to copy from.

Returns

A reference to the current Vector object.

6.11.3.17 operator=() [2/2]

```
template<unsigned int n>
Vector<n>& Vector< n >::operator= (
            Vector< n > && other )  [noexcept]
```

Move assignment operator for Vector class.

Parameters

other	The other Vector object to move from.

Returns

A reference to the current Vector object.

6.11.3.18 operator[]() [1/2]

```
template<unsigned int n>
double& Vector< n >::operator[] (
            uint32_t i )
```

Returns a reference to the i-th component of the vector.

Parameters

i	The index of the component to retrieve.

Returns

A reference to the i-th component.

6.11.3.19 operator[]() [2/2]

```
template<unsigned int n>
double Vector< n >::operator[] (
            uint32_t i ) const
```

Returns the value of the i-th component of the vector.

Parameters

i	The index of the component to retrieve.

Returns

The value of the i-th component.

6.11.3.20 random_in_unit_pos_cube()

```
template<unsigned int n>
static Vector Vector< n >::random_in_unit_pos_cube ( )  [static]
```

Returns a random vector in the unit positive cube.

Returns

A random vector in the unit positive cube.

6.11.3.21 sq_length()

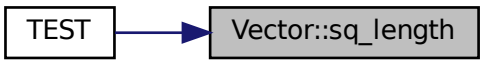
```
template<unsigned int n>
double Vector< n >::sq_length ( ) const
```

sq_length Returns the square of the length of the vector.

Returns

The square of the length of the vector.

Here is the caller graph for this function:



6.11.3.22 to_string()

```
template<unsigned int n>
std::string Vector< n >::to_string ( ) const  [inline]
```

Returns a string representation of the vector.

Returns

A string representation of the vector.

Definition at line 94 of file Vector.h.

6.11.3.23 unit()

```
template<unsigned int n>
static Vector Vector< n >::unit (
            uint32_t i )  [inline], [static]
```

Returns a vector with 1 on the i-dimension and 0 for all the other components.

Returns

A zero vector.

Definition at line 69 of file Vector.h.

Here is the call graph for this function:

6.11.3.24 zero()

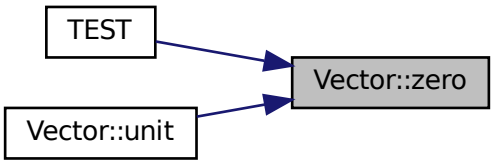
```
template<unsigned int n>
static Vector Vector< n >::zero ( )  [static]
```

Returns a zero vector.

Returns

A zero vector.

Here is the caller graph for this function:



6.11.4 Friends And Related Function Documentation

6.11.4.1 operator"!=

```
template<unsigned int n>
bool operator!= (
            const Vector< n > & a,
            const Vector< n > & b )  [friend]
```

operator!= Compares two vectors for inequality.

Parameters

a	The first vector.
b	The second vector.

Returns

`true` if the vectors are not equal, `false` otherwise.

Definition at line 197 of file Vector.h.

6.11.4.2 operator<<

```
template<unsigned int n>
std::ostream& operator<< (
            std::ostream & out,
            const Vector< n > & v )  [friend]
```

Overloaded output stream operator for printing the vector.

Parameters

out	The output stream.
v	The vector to print.

Returns

The output stream.

Definition at line 83 of file Vector.h.

6.11.4.3 operator==

```
template<unsigned int n>
bool operator== (
            const Vector< n > & a,
            const Vector< n > & b )  [friend]
```

operator== Compares two vectors for equality.

Parameters

a	The first vector.
b	The second vector.

Returns

`true` if the vectors are equal, `false` otherwise.

Definition at line 183 of file Vector.h.

6.11.5 Member Data Documentation

6.11.5.1 _components

```
template<unsigned int n>
double Vector< n >::_components[n]
```

Components of the vector.

Definition at line 12 of file Vector.h.

The documentation for this class was generated from the following file:

- include/Vector.h

Chapter 7

File Documentation

7.1 build/CMakeFiles/3.22.1/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define C_VERSION

Functions

- int main (int argc, char *argv[ ])

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

7.1.1 Macro Definition Documentation

7.1.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 17 of file CMakeCCompilerId.c.

7.1.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 652 of file CMakeCCompilerId.c.

7.1.1.3 C_VERSION

```
#define C_VERSION
```

Definition at line 741 of file CMakeCCompilerId.c.

7.1.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 396 of file CMakeCCompilerId.c.

7.1.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 656 of file CMakeCCompilerId.c.

7.1.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 667 of file CMakeCCompilerId.c.

7.1.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 524 of file CMakeCCompilerId.c.

7.1.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

Definition at line 417 of file CMakeCCompilerId.c.

7.1.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

Definition at line 416 of file CMakeCCompilerId.c.

7.1.2 Function Documentation

7.1.2.1 main()

```
int main (
             int argc,
             char * argv[] )
```

Definition at line 776 of file CMakeCCompilerId.c.

7.1.3 Variable Documentation

7.1.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 733 of file CMakeCCompilerId.c.

7.1.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 403 of file CMakeCCompilerId.c.

7.1.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
  "OFF"
"]"
```

Definition at line 757 of file CMakeCCompilerId.c.

7.1.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=
  "INFO" ":" "standard_default[" C_VERSION "]"
```

Definition at line 754 of file CMakeCCompilerId.c.

7.1.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 732 of file CMakeCCompilerId.c.

7.2 cmake-build-debug/CMakeFiles/3.25.2/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define C_VERSION

Functions

- int main (int argc, char *argv[ ])

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

7.2.1 Macro Definition Documentation

7.2.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 17 of file CMakeCCompilerId.c.

7.2.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 718 of file CMakeCCompilerId.c.

7.2.1.3 C_VERSION

```
#define C_VERSION
```

Definition at line 807 of file CMakeCCompilerId.c.

7.2.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 429 of file CMakeCCompilerId.c.

7.2.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 722 of file CMakeCCompilerId.c.

7.2.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
  ('0' + ((n)>>28 & 0xF)), \
  ('0' + ((n)>>24 & 0xF)), \
  ('0' + ((n)>>20 & 0xF)), \
  ('0' + ((n)>>16 & 0xF)), \
  ('0' + ((n)>>12 & 0xF)), \
  ('0' + ((n)>>8  & 0xF)), \
  ('0' + ((n)>>4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 733 of file CMakeCCompilerId.c.

7.2.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 560 of file CMakeCCompilerId.c.

7.2.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

Definition at line 450 of file CMakeCCompilerId.c.

7.2.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

Definition at line 449 of file CMakeCCompilerId.c.

7.2.2 Function Documentation

7.2.2.1 main()

```
int main (
             int argc,
             char * argv[] )
```

Definition at line 841 of file CMakeCCompilerId.c.

7.2.3 Variable Documentation

7.2.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 799 of file CMakeCCompilerId.c.

7.2.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 436 of file CMakeCCompilerId.c.

7.2.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
  "OFF"
"]"
```

Definition at line 823 of file CMakeCCompilerId.c.

7.2.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=
  "INFO" ":" "standard_default[" C_VERSION "]"
```

Definition at line 820 of file CMakeCCompilerId.c.

7.2.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 798 of file CMakeCCompilerId.c.

7.3 test/cmake-build-debug/CMakeFiles/3.25.2/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define C_VERSION

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char const ∗ info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const ∗ info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const ∗ info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char ∗ info_language_standard_default
- const char ∗ info_language_extensions_default

7.3.1 Macro Definition Documentation

7.3.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 17 of file CMakeCCompilerId.c.

7.3.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 718 of file CMakeCCompilerId.c.

7.3.1.3 C_VERSION

```
#define C_VERSION
```

Definition at line 807 of file CMakeCCompilerId.c.

7.3.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 429 of file CMakeCCompilerId.c.

7.3.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 722 of file CMakeCCompilerId.c.

7.3.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 733 of file CMakeCCompilerId.c.

7.3.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 560 of file CMakeCCompilerId.c.

7.3.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

Definition at line 450 of file CMakeCCompilerId.c.

7.3.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

Definition at line 449 of file CMakeCCompilerId.c.

7.3.2 Function Documentation

7.3.2.1 main()

```
int main (
             int argc,
             char * argv[] )
```

Definition at line 841 of file CMakeCCompilerId.c.

7.3.3 Variable Documentation

7.3.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 799 of file CMakeCCompilerId.c.

7.3.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 436 of file CMakeCCompilerId.c.

7.3.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
  "OFF"
"]"
```

Definition at line 823 of file CMakeCCompilerId.c.

7.3.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
=
  "INFO" ":" "standard_default[" C_VERSION "]"
```

Definition at line 820 of file CMakeCCompilerId.c.

7.3.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 798 of file CMakeCCompilerId.c.

7.4 build/CMakeFiles/3.22.1/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX_STD __cplusplus

Functions

- int main (int argc, char *argv[])

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

7.4.1 Macro Definition Documentation

7.4.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 11 of file CMakeCXXCompilerId.cpp.

7.4.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 637 of file CMakeCXXCompilerId.cpp.

7.4.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 381 of file CMakeCXXCompilerId.cpp.

7.4.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 735 of file CMakeCXXCompilerId.cpp.

7.4.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 641 of file CMakeCXXCompilerId.cpp.

7.4.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 652 of file CMakeCXXCompilerId.cpp.

7.4.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 509 of file CMakeCXXCompilerId.cpp.

7.4.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

Definition at line 402 of file CMakeCXXCompilerId.cpp.

7.4.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

Definition at line 401 of file CMakeCXXCompilerId.cpp.

7.4.2 Function Documentation

7.4.2.1 main()

```
int main (
             int argc,
             char * argv[] )
```

Definition at line 767 of file CMakeCXXCompilerId.cpp.

7.4.3 Variable Documentation

7.4.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 718 of file CMakeCXXCompilerId.cpp.

7.4.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 388 of file CMakeCXXCompilerId.cpp.

7.4.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
  "OFF"
"]"
```

Definition at line 754 of file CMakeCXXCompilerId.cpp.

7.4.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["
  "98"
"]"
```

Definition at line 738 of file CMakeCXXCompilerId.cpp.

7.4.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 717 of file CMakeCXXCompilerId.cpp.

7.5 cmake-build-debug/CMakeFiles/3.25.2/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX_STD __cplusplus

Functions

- int main (int argc, char *argv[])

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_standard_default
- const char * info_language_extensions_default

7.5.1 Macro Definition Documentation

7.5.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 11 of file CMakeCXXCompilerId.cpp.

7.5.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 703 of file CMakeCXXCompilerId.cpp.

7.5.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 414 of file CMakeCXXCompilerId.cpp.

7.5.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 801 of file CMakeCXXCompilerId.cpp.

7.5.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 707 of file CMakeCXXCompilerId.cpp.

7.5.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 718 of file CMakeCXXCompilerId.cpp.

7.5.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 545 of file CMakeCXXCompilerId.cpp.

7.5.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

Definition at line 435 of file CMakeCXXCompilerId.cpp.

7.5.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

Definition at line 434 of file CMakeCXXCompilerId.cpp.

7.5.2 Function Documentation

7.5.2.1 main()

```
int main (
             int argc,
             char * argv[] )
```

Definition at line 832 of file CMakeCXXCompilerId.cpp.

7.5.3 Variable Documentation

7.5.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 784 of file CMakeCXXCompilerId.cpp.

7.5.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 421 of file CMakeCXXCompilerId.cpp.

7.5.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
  "OFF"
"]"
```

Definition at line 820 of file CMakeCXXCompilerId.cpp.

7.5.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["
  "98"
"]"
```

Definition at line 804 of file CMakeCXXCompilerId.cpp.

7.5.3.5 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 783 of file CMakeCXXCompilerId.cpp.

7.6 test/cmake-build-debug/CMakeFiles/3.25.2/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define __has_include(x) 0
- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX_STD __cplusplus

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char const ∗ info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const ∗ info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const ∗ info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char ∗ info_language_standard_default
- const char ∗ info_language_extensions_default

7.6.1 Macro Definition Documentation

7.6.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 11 of file CMakeCXXCompilerId.cpp.

7.6.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 703 of file CMakeCXXCompilerId.cpp.

7.6.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 414 of file CMakeCXXCompilerId.cpp.

7.6.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 801 of file CMakeCXXCompilerId.cpp.

7.6.1.5 DEC

```
#define DEC(
              n )
```

Value:

```
  ('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 707 of file CMakeCXXCompilerId.cpp.

7.6.1.6 HEX

```
#define HEX(
              n )
```

Value:

```
  ('0' + ((n)»28 & 0xF)), \
  ('0' + ((n)»24 & 0xF)), \
  ('0' + ((n)»20 & 0xF)), \
  ('0' + ((n)»16 & 0xF)), \
  ('0' + ((n)»12 & 0xF)), \
  ('0' + ((n)»8  & 0xF)), \
  ('0' + ((n)»4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 718 of file CMakeCXXCompilerId.cpp.

7.6.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 545 of file CMakeCXXCompilerId.cpp.

7.6.1.8 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

Definition at line 435 of file CMakeCXXCompilerId.cpp.

7.6.1.9 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

Definition at line 434 of file CMakeCXXCompilerId.cpp.

7.6.2 Function Documentation

7.6.2.1 main()

```
int main (
             int argc,
             char * argv[] )
```

Definition at line 832 of file CMakeCXXCompilerId.cpp.

7.6.3 Variable Documentation

7.6.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 784 of file CMakeCXXCompilerId.cpp.

7.6.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 421 of file CMakeCXXCompilerId.cpp.

7.6.3.3 info_language_extensions_default

```
const char* info_language_extensions_default
```

Initial value:

```
= "INFO" ":" "extensions_default["
  "OFF"
"]"
```

Definition at line 820 of file CMakeCXXCompilerId.cpp.

7.6.3.4 info_language_standard_default

```
const char* info_language_standard_default
```

Initial value:

```
= "INFO" ":" "standard_default["
  "98"
"]"
```

Definition at line 804 of file CMakeCXXCompilerId.cpp.

7.6.3.5 info_platform

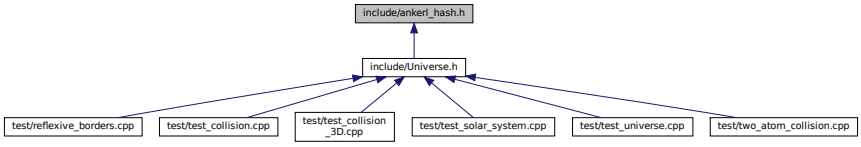
```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 783 of file CMakeCXXCompilerId.cpp.

7.7 demo/demo.cpp File Reference

7.8 include/ankerl_hash.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define ANKERL_UNORDERED_DENSE_VERSION_MAJOR 4
- #define ANKERL_UNORDERED_DENSE_VERSION_MINOR 0
- #define ANKERL_UNORDERED_DENSE_VERSION_PATCH 0
- #define ANKERL_UNORDERED_DENSE_VERSION_CONCAT1(major, minor, patch) v##major##_↩ ##minor##_##patch
- #define ANKERL_UNORDERED_DENSE_VERSION_CONCAT(major, minor, patch) ANKERL_UNORDERED_DENSE_VERS minor, patch)
- #define ANKERL_UNORDERED_DENSE_NAMESPACE
- #define ANKERL_UNORDERED_DENSE_CPP_VERSION __cplusplus
- #define ANKERL_UNORDERED_DENSE_HAS_EXCEPTIONS() 0
- #define ANKERL_UNORDERED_DENSE_NOINLINE __attribute__((noinline))

7.8.1 Macro Definition Documentation

7.8.1.1 ANKERL_UNORDERED_DENSE_CPP_VERSION

```
#define ANKERL_UNORDERED_DENSE_CPP_VERSION __cplusplus
```

Definition at line 50 of file ankerl_hash.h.

7.8.1.2 ANKERL_UNORDERED_DENSE_HAS_EXCEPTIONS

```
#define ANKERL_UNORDERED_DENSE_HAS_EXCEPTIONS( ) 0
```

Definition at line 65 of file ankerl_hash.h.

7.8.1.3 ANKERL_UNORDERED_DENSE_NAMESPACE

```
#define ANKERL_UNORDERED_DENSE_NAMESPACE
```

Value:

```
ANKERL_UNORDERED_DENSE_VERSION_CONCAT( \
    ANKERL_UNORDERED_DENSE_VERSION_MAJOR, ANKERL_UNORDERED_DENSE_VERSION_MINOR,
   ANKERL_UNORDERED_DENSE_VERSION_PATCH)
```

Definition at line 43 of file ankerl_hash.h.

7.8.1.4 ANKERL_UNORDERED_DENSE_NOINLINE

```
#define ANKERL_UNORDERED_DENSE_NOINLINE __attribute__((noinline))
```

Definition at line 70 of file ankerl_hash.h.

7.8.1.5 ANKERL_UNORDERED_DENSE_VERSION_CONCAT

```
#define ANKERL_UNORDERED_DENSE_VERSION_CONCAT(
              major,
              minor,
              patch ) ANKERL_UNORDERED_DENSE_VERSION_CONCAT1(major, minor, patch)
```

Definition at line 42 of file ankerl_hash.h.

7.8.1.6 ANKERL_UNORDERED_DENSE_VERSION_CONCAT1

```
#define ANKERL_UNORDERED_DENSE_VERSION_CONCAT1(
              major,
              minor,
              patch ) v##major##_##minor##_##patch
```

Definition at line 40 of file ankerl_hash.h.

7.8.1.7 ANKERL_UNORDERED_DENSE_VERSION_MAJOR

```
#define ANKERL_UNORDERED_DENSE_VERSION_MAJOR 4
```

Definition at line 33 of file ankerl_hash.h.

7.8.1.8 ANKERL_UNORDERED_DENSE_VERSION_MINOR

```
#define ANKERL_UNORDERED_DENSE_VERSION_MINOR 0
```

Definition at line 34 of file ankerl_hash.h.

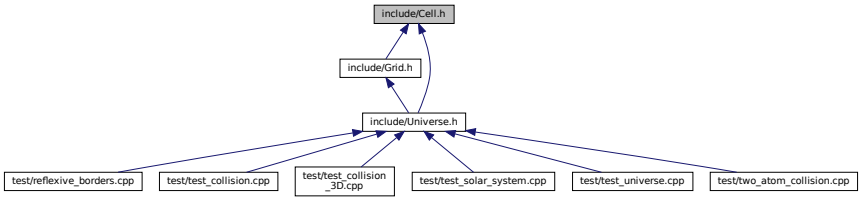
7.8.1.9 ANKERL_UNORDERED_DENSE_VERSION_PATCH

```
#define ANKERL_UNORDERED_DENSE_VERSION_PATCH 0
```

Definition at line 35 of file ankerl_hash.h.

7.9 include/Cell.h File Reference

This graph shows which files directly or indirectly include this file:



Classes

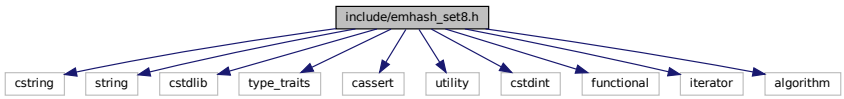
- class Cell

  Class representing a cell in the grid.

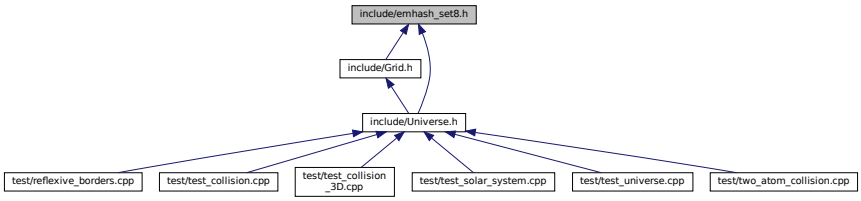
7.10 include/emhash_set8.h File Reference

```
#include <cstring>
#include <string>
#include <cstdlib>
#include <type_traits>
#include <cassert>
#include <utility>
#include <cstdint>
#include <functional>
#include <iterator>
#include <algorithm>
```

Include dependency graph for emhash_set8.h:



This graph shows which files directly or indirectly include this file:



Classes

- class emhash8::HashSet< KeyT, HashT, EqT >

  A cache-friendly hash table with open addressing, linear/quadratic probing and power-of-two capacity.
- struct emhash8::HashSet< KeyT, HashT, EqT >::Index
- class emhash8::HashSet< KeyT, HashT, EqT >::iterator
- class emhash8::HashSet< KeyT, HashT, EqT >::const_iterator

Namespaces

- emhash8

Macros

- #define EMH_LIKELY(condition) (condition)
- #define EMH_UNLIKELY(condition) (condition)
- #define EMH_KEY(p, n) p[n]
- #define EMH_INDEX(i, n) i[n]
- #define EMH_BUCKET(i, n) i[n].bucket
- #define EMH_HSLOT(i, n) i[n].slot
- #define EMH_SLOT(i, n) (i[n].slot & _mask)
- #define EMH_PREVET(i, n) i[n].slot
- #define EMH_KEYMASK(key, mask) ((size_type)(key >> 0) & ~mask)
- #define EMH_EQHASH(n, key_hash) (EMH_KEYMASK(key_hash, _mask) == (_index[n].slot & ~_mask))
- #define EMH_NEW(key, bucket, key_hash) new(_pairs + _num_filled) value_type(key); _index[bucket] = {bucket, _num_filled++ | EMH_KEYMASK(key_hash, _mask)}
- #define EMH_EMPTY(i, n) (0 > (int)i[n].bucket)

Variables

- constexpr uint32_t emhash8::INACTIVE = 0xAAAAAAAA
- constexpr uint32_t emhash8::END = 0-0x1u
- constexpr uint32_t emhash8::EAD = 2
- constexpr static float emhash8::EMH_DEFAULT_LOAD_FACTOR = 0.80f
- constexpr static uint32_t emhash8::EMH_CACHE_LINE_SIZE = 64

7.10.1 Macro Definition Documentation

7.10.1.1 EMH_BUCKET

```
#define EMH_BUCKET(
              i,
              n ) i[n].bucket
```

Definition at line 59 of file emhash_set8.h.

7.10.1.2 EMH_EMPTY

```
#define EMH_EMPTY(
              i,
              n ) (0 > (int)i[n].bucket)
```

Definition at line 68 of file emhash_set8.h.

7.10.1.3 EMH_EQHASH

```
#define EMH_EQHASH(
              n,
              key_hash ) (EMH_KEYMASK(key_hash, _mask) == (_index[n].slot & ~_mask))
```

Definition at line 65 of file emhash_set8.h.

7.10.1.4 EMH_HSLOT

```
#define EMH_HSLOT(
              i,
              n ) i[n].slot
```

Definition at line 60 of file emhash_set8.h.

7.10.1.5 EMH_INDEX

```
#define EMH_INDEX(
              i,
              n ) i[n]
```

Definition at line 58 of file emhash_set8.h.

7.10.1.6 EMH_KEY

```
#define EMH_KEY(
              p,
              n ) p[n]
```

Definition at line 56 of file emhash_set8.h.

7.10.1.7 EMH_KEYMASK

```
#define EMH_KEYMASK(
              key,
              mask ) ((size_type)(key >> 0) & ~mask)
```

Definition at line 64 of file emhash_set8.h.

7.10.1.8 EMH_LIKELY

```
#define EMH_LIKELY(
              condition ) (condition)
```

Definition at line 52 of file emhash_set8.h.

7.10.1.9 EMH_NEW

```
#define EMH_NEW(
              key,
              bucket,
              key_hash ) new(_pairs + _num_filled) value_type(key); _index[bucket] = {bucket,
_num_filled++ | EMH_KEYMASK(key_hash, _mask)}
```

Definition at line 66 of file emhash_set8.h.

7.10.1.10 EMH_PREVET

```
#define EMH_PREVET(
              i,
              n ) i[n].slot
```

Definition at line 62 of file emhash_set8.h.

7.10.1.11 EMH_SLOT

```
#define EMH_SLOT(
              i,
              n ) (i[n].slot & _mask)
```

Definition at line 61 of file emhash_set8.h.

7.10.1.12 EMH_UNLIKELY

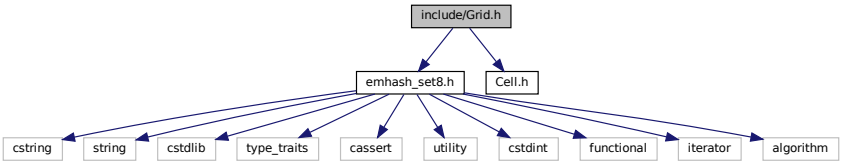
```
#define EMH_UNLIKELY(
              condition ) (condition)
```

Definition at line 53 of file emhash_set8.h.

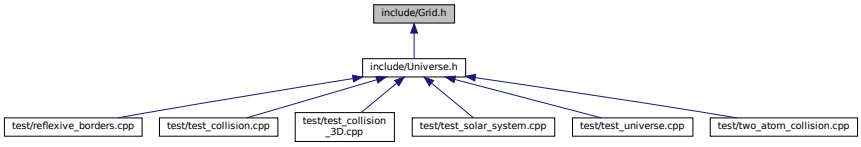
7.11 include/Grid.h File Reference

```
#include "emhash_set8.h"
#include "Cell.h"
```

Include dependency graph for Grid.h:



This graph shows which files directly or indirectly include this file:



Classes

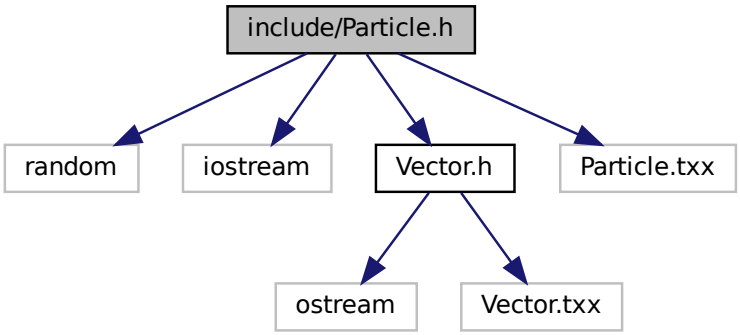
- struct Grid< n >

7.12 include/Particle.h File Reference

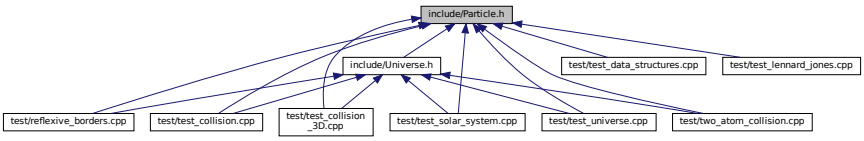
Defines a Particle class representing a particle in an n-dimensional world, described by its position, speed, mass, and category.

```
#include <random>
#include <iostream>
#include "Vector.h"
#include "Particle.txx"
```

Include dependency graph for Particle.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Particle< n >

  Represents a particle in an n-dimensional world, described by its position, speed, mass, and category.

Enumerations

- enum Category { ELECTRON , PROTON , NEUTRON }

  Represents the type of a particle.

7.12.1 Detailed Description

Defines a Particle class representing a particle in an n-dimensional world, described by its position, speed, mass, and category.

Author

helen

Date

23/02/23

7.12.2 Enumeration Type Documentation

7.12.2.1 Category

```
enum Category
```

Represents the type of a particle.

Enumerator

ELECTRON	Represents an electron particle.
PROTON	Represents a proton particle.
NEUTRON	Represents a neutron particle.

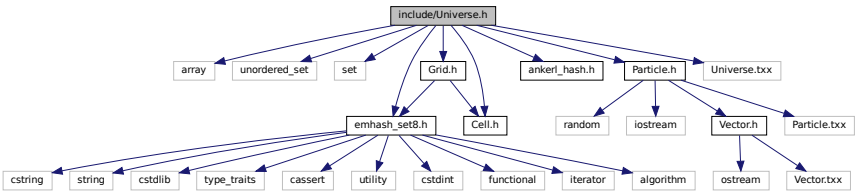
Definition at line 19 of file Particle.h.

7.13 include/Universe.h File Reference

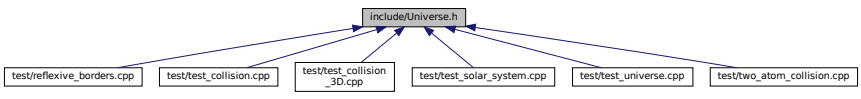
Definition of the Universe class.

```
#include <array>
#include <unordered_set>
#include <set>
#include "emhash_set8.h"
#include "ankerl_hash.h"
#include "Particle.h"
#include "Cell.h"
#include "Grid.h"
#include "Universe.txx"
```

Include dependency graph for Universe.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct SimulationSettings
- struct SimulationConstraints< n >
- class Universe< n >

  Class representing the simulation universe.

Enumerations

- enum BoundaryBehaviour { Reflexive , ReflexivePotential , Absorption , Periodic }

7.13.1 Detailed Description

Definition of the Universe class.

7.13.2 Enumeration Type Documentation

7.13.2.1 BoundaryBehaviour

```
enum BoundaryBehaviour
```

Enumerator

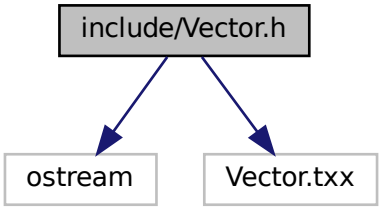
Reflexive	
ReflexivePotential	
Absorption	
Periodic	

Definition at line 20 of file Universe.h.

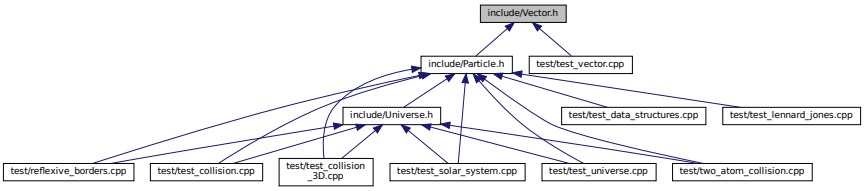
7.14 include/Vector.h File Reference

```
#include <ostream>
#include "Vector.txx"
```

Include dependency graph for Vector.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Vector< n >

Functions

- template<unsigned int n>
  Vector< n > operator+ (Vector< n > a, const Vector< n > &b)

  Adds two vectors component-wise.
- template<unsigned int n>
  Vector< n > operator- (Vector< n > a, const Vector< n > &b)

  Subtracts two vectors component-wise.
- template<unsigned int n>
  Vector< n > operator* (Vector< n > a, const Vector< n > &b)

  Multiplies two vectors component-wise.
- template<unsigned int n>
  Vector< n > operator* (double a, Vector< n > b)

  Multiplies a vector by a scalar.
- template<unsigned int n>
  Vector< n > operator* (Vector< n > a, double b)

  Multiplies a vector by a scalar.
- template<unsigned int n>
  Vector< n > operator/ (Vector< n > a, double b)

  Divides a vector by a scalar.

- template<unsigned int n>
  Vector< n > operator% (Vector< n > a, const Vector< n > &b)

  Apply modulo b[i] to each component a[i] of this vector.
- template<unsigned int n>
  Vector< n > operator% (Vector< n > a, double b)

  Apply modulo b to each component a[i] of this vector.
- template<unsigned int n>
  Vector< n > operator/ (Vector< n > a, const Vector< n > &b)

  Divides a vector by a scalar.

7.14.1 Function Documentation

7.14.1.1 operator%() [1/2]

```
template<unsigned int n>
Vector<n> operator% (
            Vector< n > a,
            const Vector< n > & b )
```

Apply modulo b[i] to each component a[i] of this vector.

Template Parameters

n	The number of components in the vector.

Parameters

a	The vector to divide.
b	The modulo.

Returns

The vector resulting from the modulo.

7.14.1.2 operator%() [2/2]

```
template<unsigned int n>
Vector<n> operator% (
            Vector< n > a,
            double b )
```

Apply modulo b to each component a[i] of this vector.

Template Parameters

n	The number of components in the vector.

Parameters

a	The vector to divide.
b	The scalar

Returns

The vector resulting from the modulo.

7.14.1.3 operator∗() [1/3]

```
template<unsigned int n>
Vector<n> operator* (
            double a,
            Vector< n > b )
```

Multiplies a vector by a scalar.

Template Parameters

n	The number of components in the vector.

Parameters

a	The scalar to multiply the vector by.
b	The vector to multiply.

Returns

The vector resulting from multiplying the input vector by the scalar.

7.14.1.4 operator∗() [2/3]

```
template<unsigned int n>
Vector<n> operator* (
            Vector< n > a,
            const Vector< n > & b )
```

Multiplies two vectors component-wise.

Template Parameters

n	The number of components in the vectors.

Parameters

a	The first vector to multiply.
b	The second vector to multiply.

Returns

The vector resulting from multiplying the two input vectors component-wise.

7.14.1.5 operator*() [3/3]

```
template<unsigned int n>
Vector<n> operator* (
            Vector< n > a,
            double b )
```

Multiplies a vector by a scalar.

Template Parameters

n	The number of components in the vector.

Parameters

a	The vector to multiply.
b	The scalar to multiply the vector by.

Returns

The vector resulting from multiplying the input vector by the scalar.

7.14.1.6 operator+()

```
template<unsigned int n>
Vector<n> operator+ (
            Vector< n > a,
            const Vector< n > & b )
```

Adds two vectors component-wise.

Template Parameters

n	The number of components in the vectors.

Parameters

a	The first vector to add.
b	The second vector to add.

Returns

The vector resulting from adding the two input vectors component-wise.

7.14.1.7 operator-()

```
template<unsigned int n>
Vector<n> operator- (
            Vector< n > a,
            const Vector< n > & b )
```

Subtracts two vectors component-wise.

Template Parameters

n	The number of components in the vectors.

Parameters

a	The vector to subtract from.
b	The vector to subtract.

Returns

The vector resulting from subtracting the second vector from the first component-wise.

7.14.1.8 operator/() [1/2]

```
template<unsigned int n>
Vector<n> operator/ (
            Vector< n > a,
            const Vector< n > & b )
```

Divides a vector by a scalar.

Template Parameters

n	The number of components in the vector.

Parameters

a	The vector to divide.
b	The scalar to divide the vector by.

Returns

The vector resulting from dividing the input vector by the scalar.

7.14.1.9 operator/() [2/2]

```
template<unsigned int n>
Vector<n> operator/ (
            Vector< n > a,
            double b )
```

Divides a vector by a scalar.

Template Parameters

n	The number of components in the vector.

Parameters

a	The vector to divide.
b	The scalar to divide the vector by.

Returns

The vector resulting from dividing the input vector by the scalar.

7.15 README.md File Reference

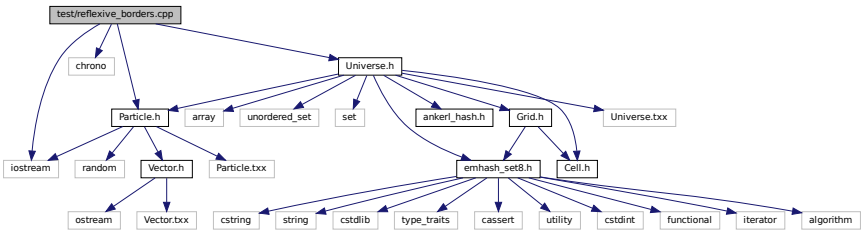
7.16 test/cmake-build-debug/CMakeFiles/TestVectors.dir/test_↩ vector.cpp.o.d File Reference

7.17 test/reflexive_borders.cpp File Reference

```
#include <iostream>
#include <chrono>
```

```
#include "Particle.h"
#include "Universe.h"
```

Include dependency graph for reflexive_borders.cpp:



Functions

- int main ()

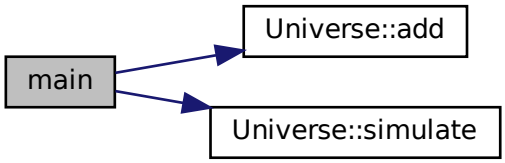
7.17.1 Function Documentation

7.17.1.1 main()

```
int main ( )
```

Definition at line 9 of file reflexive_borders.cpp.

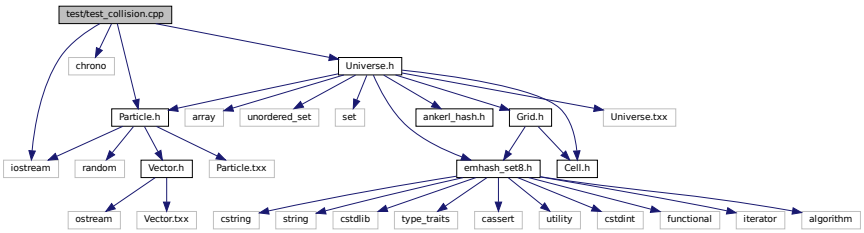
Here is the call graph for this function:



7.18 test/test_collision.cpp File Reference

```
#include <iostream>
#include <chrono>
#include "Particle.h"
#include "Universe.h"
```

Include dependency graph for test_collision.cpp:



Functions

- int main ()

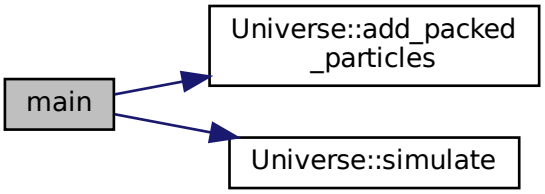
7.18.1 Function Documentation

7.18.1.1 main()

```
int main ( )
```

Definition at line 6 of file test_collision.cpp.

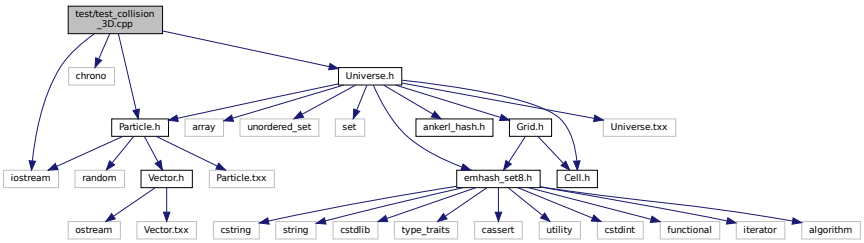
Here is the call graph for this function:



7.19 test/test_collision_3D.cpp File Reference

```
#include <iostream>
#include <chrono>
#include "Particle.h"
#include "Universe.h"
```

Include dependency graph for test_collision_3D.cpp:



Functions

- int main ()

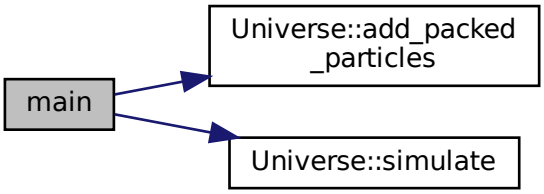
7.19.1 Function Documentation

7.19.1.1 main()

```
int main ( )
```

Definition at line 6 of file test_collision_3D.cpp.

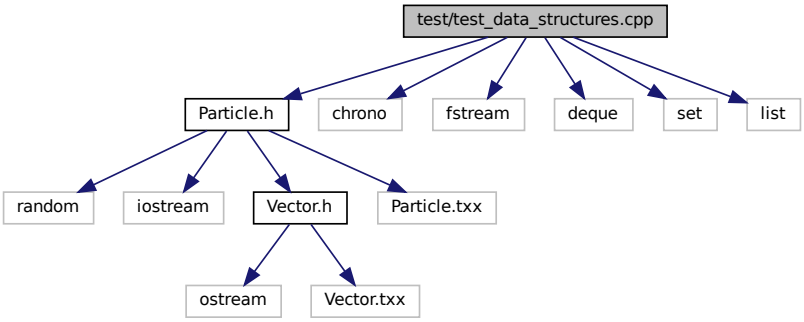
Here is the call graph for this function:



7.20 test/test_data_structures.cpp File Reference

```
#include "Particle.h"
#include <chrono>
#include <fstream>
#include <deque>
#include <set>
#include <list>
```

Include dependency graph for test_data_structures.cpp:



Functions

- auto new_random_vec3 (std::mt19937 &gen, std::uniform_real_distribution< double > &distribution) -> Vector< 3 >
- auto new_random_double (std::mt19937 &gen, std::uniform_real_distribution< double > &distribution) -> double
- std::list< Particle< 3 > > generate_particles_list (uint32_t nb_particles)
- std::deque< Particle< 3 > > generate_particles_queue (uint32_t nb_particles)
- std::set< Particle< 3 > > generate_particles_set (uint32_t nb_particles)
- std::vector< Particle< 3 > > generate_particles_vector (uint32_t nb_particles)
- void run (std::list< uint32_t > particles_sizes)
- void data_structures_comparison ()
- int main ()

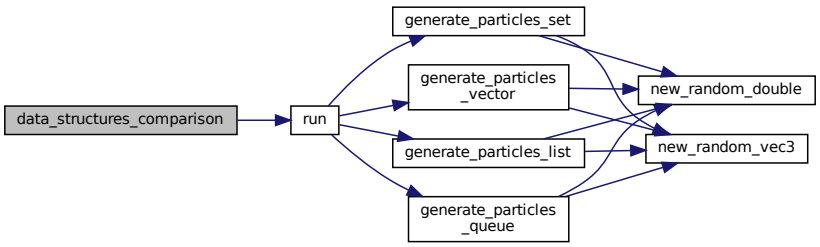
7.20.1 Function Documentation

7.20.1.1 data_structures_comparison()

```
void data_structures_comparison ( )
```

Definition at line 121 of file test_data_structures.cpp.

Here is the call graph for this function:



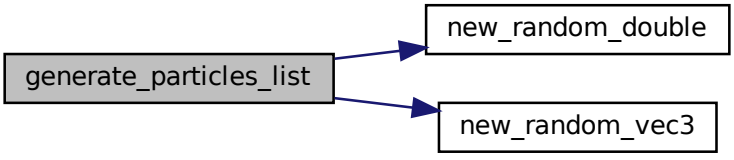
Here is the caller graph for this function:

7.20.1.2 generate_particles_list()

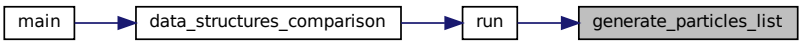
```
std::list<Particle<3> > generate_particles_list (
            uint32_t nb_particles )
```

Definition at line 26 of file test_data_structures.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

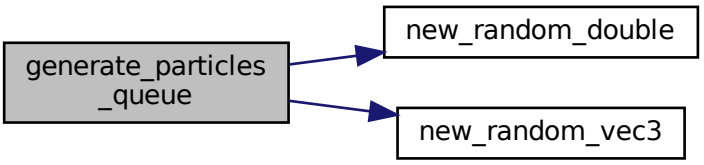


7.20.1.3 generate_particles_queue()

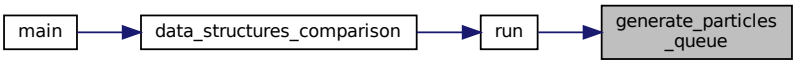
```
std::deque<Particle<3> > generate_particles_queue (
            uint32_t nb_particles )
```

Definition at line 43 of file test_data_structures.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

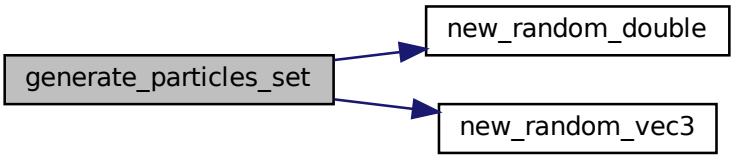


7.20.1.4 generate_particles_set()

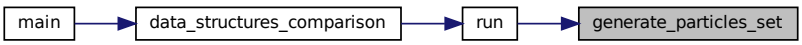
```
std::set<Particle<3> > generate_particles_set (
            uint32_t nb_particles )
```

Definition at line 59 of file test_data_structures.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

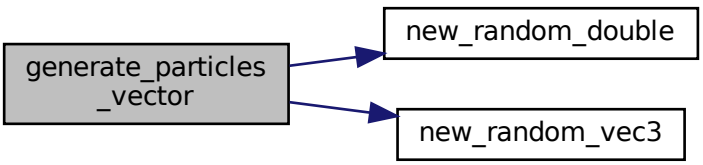


7.20.1.5 generate_particles_vector()

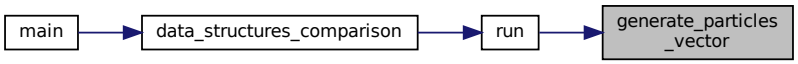
```
std::vector<Particle<3> > generate_particles_vector (
            uint32_t nb_particles )
```

Definition at line 74 of file test_data_structures.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

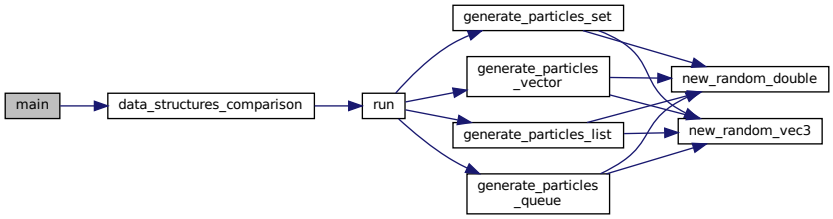


7.20.1.6 main()

```
int main ( )
```

Definition at line 126 of file test_data_structures.cpp.

Here is the call graph for this function:

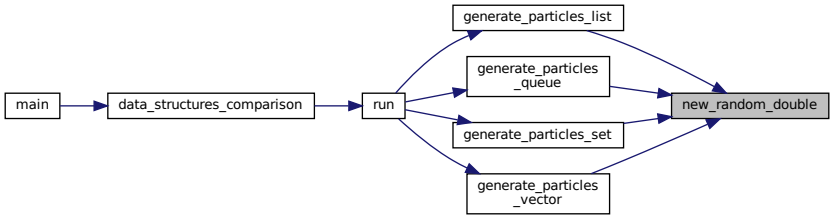


7.20.1.7 new_random_double()

```
auto new_random_double (
            std::mt19937 & gen,
            std::uniform_real_distribution< double > & distribution ) -> double
```

Definition at line 21 of file test_data_structures.cpp.

Here is the caller graph for this function:



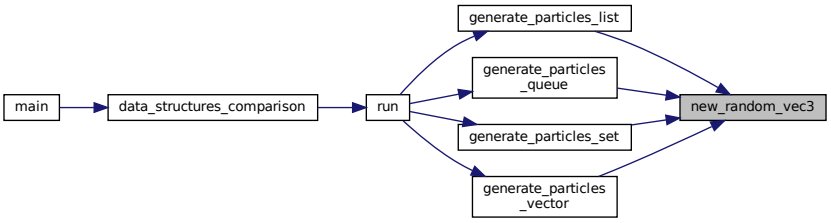
7.20.1.8 new_random_vec3()

```
auto new_random_vec3 (
            std::mt19937 & gen,
            std::uniform_real_distribution< double > & distribution ) -> Vector<3>
```

Test that compares the different data structures (set, queue, vector, list) which can be used to store Particles. We can see from the output that the vector structure is the most efficient, followed by the queue and the list, then the set. Therefore, we used a vector to store our particles in our Universe class. This test answers Part 2 of Lab 2.

Definition at line 17 of file test_data_structures.cpp.

Here is the caller graph for this function:

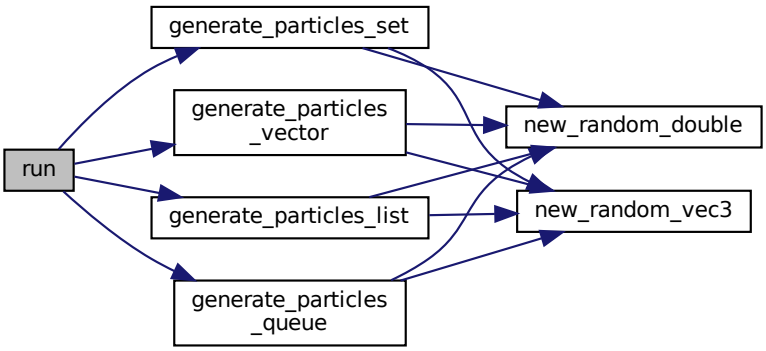


7.20.1.9 run()

```
void run (
            std::list< uint32_t > particles_sizes )
```

Definition at line 91 of file test_data_structures.cpp.

Here is the call graph for this function:

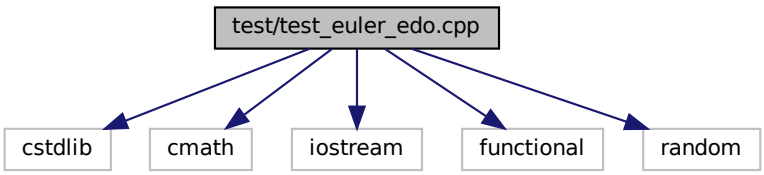


Here is the caller graph for this function:

7.21 test/test_euler_edo.cpp File Reference

```
#include <cstdlib>
#include <cmath>
#include <iostream>
#include <functional>
#include <random>
```

Include dependency graph for test_euler_edo.cpp:



Functions

- double ** initialization (int n)

  Routine d'initialization qui permet d'allouer la mémoire pour une matrice carrée de taille n et renvoie un pointer vers la matrice allouée.
- double * fill_vectors (double *vec, int n)

  Intitialise un vecteur avec des valeurs aléatoires comprises entre dans l'intervalle [-10;10].
- void print_matrix (double **matrix, int n)

  Affiche les éléments d'une matrice de taille n.
- double trace (double **matrix, int n)

  Calcule la trace d'une matrice.
- void compute_trace ()

  Driver principal pour le calcul de la tace d'une matrice.
- double phi (double x, double u_x)
- double phi2 (double x, double u_x)
- vector< double > euler_explicit (int iter_count, double initial_value, const function< double(double, double)> &phi)
- vector< double > euler_implicit (int iter_count, double initial_value, const function< double(double, double)> &phi)
- void run_euler_methods (int iter_count, double initial_value, const function< double(double, double)> &phi)
- int main ()

7.21.1 Function Documentation

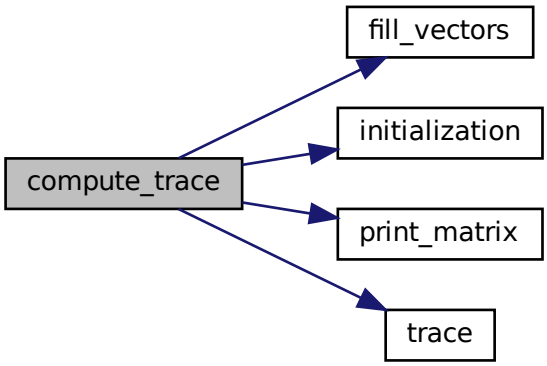
7.21.1.1 compute_trace()

```
void compute_trace ( )
```

Driver principal pour le calcul de la tace d'une matrice.

Definition at line 30 of file test_euler_edo.cpp.

Here is the call graph for this function:

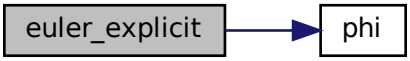


7.21.1.2 euler_explicit()

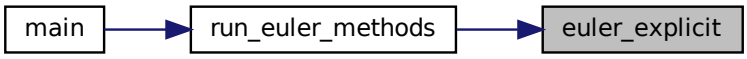
```
vector<double> euler_explicit (
            int iter_count,
            double initial_value,
            const function< double(double, double)> & phi )
```

Definition at line 111 of file test_euler_edo.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:

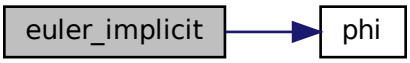


7.21.1.3 euler_implicit()

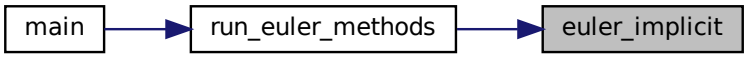
```
vector<double> euler_implicit (
            int iter_count,
            double initial_value,
            const function< double(double, double)> & phi )
```

Definition at line 121 of file test_euler_edo.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.1.4 fill_vectors()

```
double * fill_vectors (
            double * vec,
            int n )
```

Intitialise un vecteur avec des valeurs aléatoires comprises entre dans l'intervalle [-10;10].

Parameters

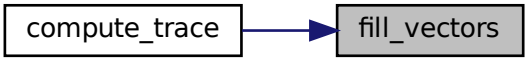
in	*vec*	est le vecteur à initialiser
in	*n*	est la taille du vecteur à initialiser

Returns

Renvoie le vecteur initialisé.

Definition at line 67 of file test_euler_edo.cpp.

Here is the caller graph for this function:



7.21.1.5 initialization()

```
double ** initialization (
             int n )
```

Routine d'initialization qui permet d'allouer la mémoire pour une matrice carrée de taille n et renvoie un pointer vers la matrice allouée.

Parameters

in	*n*	est la taille souhaitée de la matrice

Returns

Renvoie le pointeur vers la matrice allouée

Definition at line 52 of file test_euler_edo.cpp.

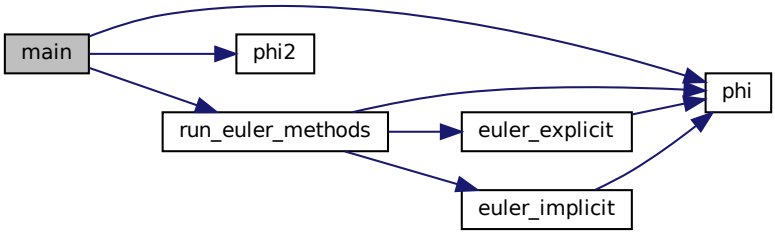
Here is the caller graph for this function:

7.21.1.6 main()

```
int main ( )
```

Definition at line 148 of file test_euler_edo.cpp.

Here is the call graph for this function:

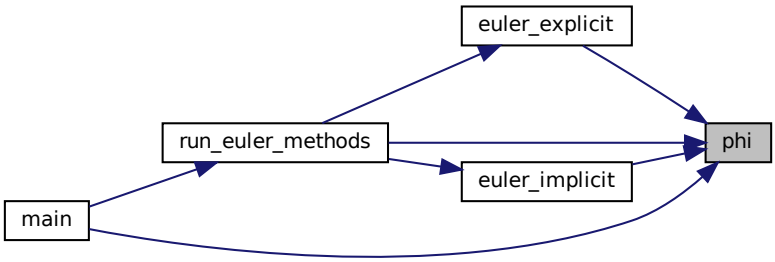


7.21.1.7 phi()

```
double phi (
            double x,
            double u_x )
```

Definition at line 102 of file test_euler_edo.cpp.

Here is the caller graph for this function:



7.21.1.8 phi2()

```
double phi2 (
            double x,
            double u_x )
```

Definition at line 106 of file test_euler_edo.cpp.

Here is the caller graph for this function:

7.21.1.9 print_matrix()

```
void print_matrix (
            double ** matrix,
            int n )
```

Affiche les éléments d'une matrice de taille n.

Parameters

in	*matrix*	est la matrice à afficher
in	*n*	est la taille de la matrice à afficher

Definition at line 78 of file test_euler_edo.cpp.

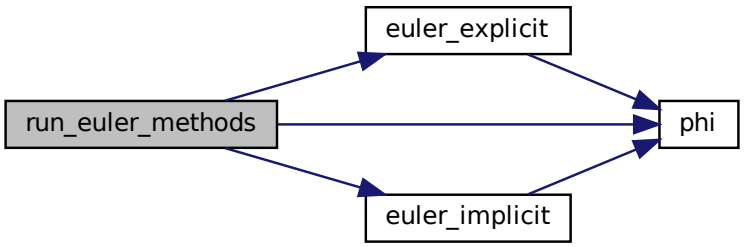
Here is the caller graph for this function:

7.21.1.10 run_euler_methods()

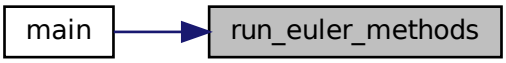
```
void run_euler_methods (
            int iter_count,
            double initial_value,
            const function< double(double, double)> & phi )
```

Definition at line 132 of file test_euler_edo.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.1.11 trace()

```
double trace (
            double ** matrix,
            int n )
```

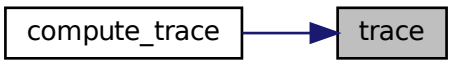
Calcule la trace d'une matrice.

Parameters

in	*matrix*	est la matrice dont on souhaite connaître la trace.
in		

Definition at line 92 of file test_euler_edo.cpp.

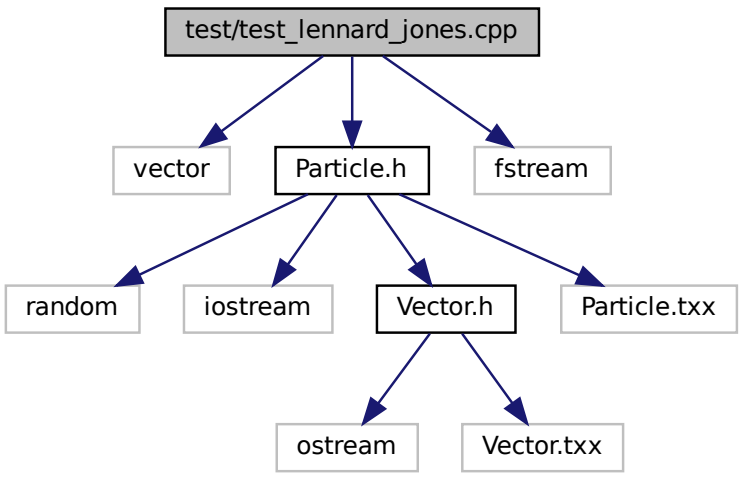
Here is the caller graph for this function:



7.22 test/test_lennard_jones.cpp File Reference

```
#include <vector>
#include <Particle.h>
#include <fstream>
```

Include dependency graph for test_lennard_jones.cpp:



Functions

- auto new_random_vec3 (std::mt19937 &gen, std::uniform_real_distribution< double > &distribution) -> Vector< 3 >
- auto new_random_double (std::mt19937 &gen, std::uniform_real_distribution< double > &distribution) -> double
- std::vector< Particle< 3 > > generate_particles_vector (uint32_t nb_particles)
- void lennard_jones_potential (std::vector< Particle< 3 >> particles)
- int main ()

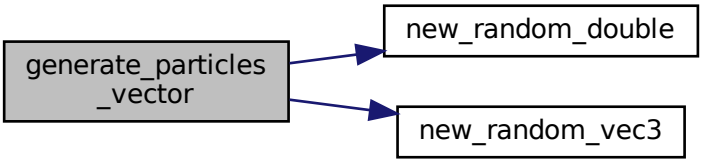
7.22.1 Function Documentation

7.22.1.1 generate_particles_vector()

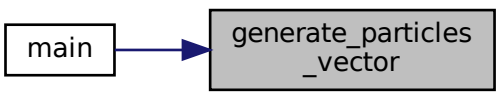
```
std::vector<Particle<3> > generate_particles_vector (
            uint32_t nb_particles )
```

Definition at line 30 of file test_lennard_jones.cpp.

Here is the call graph for this function:



Here is the caller graph for this function:



7.22.1.2 lennard_jones_potential()

```
void lennard_jones_potential (
            std::vector< Particle< 3 >> particles )
```

Definition at line 48 of file test_lennard_jones.cpp.

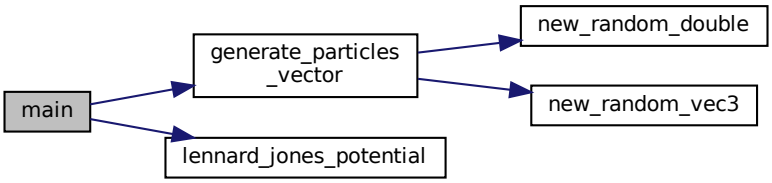
Here is the caller graph for this function:

7.22.1.3 main()

```
int main ( )
```

Definition at line 77 of file test_lennard_jones.cpp.

Here is the call graph for this function:

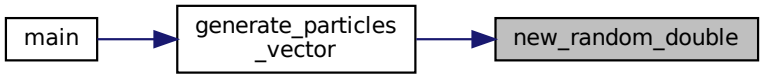


7.22.1.4 new_random_double()

```
auto new_random_double (
            std::mt19937 & gen,
            std::uniform_real_distribution< double > & distribution ) -> double
```

Definition at line 26 of file test_lennard_jones.cpp.

Here is the caller graph for this function:



7.22.1.5 new_random_vec3()

```
auto new_random_vec3 (
            std::mt19937 & gen,
            std::uniform_real_distribution< double > & distribution ) -> Vector<3>
```

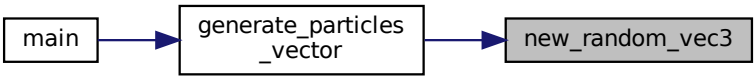
Test that displays Lennard-Jones potential for a system composed of 2 particles. This test answers to the first question of Lab 4.

Here are the commands to execute in gnuplot (each command needs to be on separate lines) after running the test to display the actual graph: set xlabel "r" set ylabel "U" set xzeroaxis lt -1 lw 1 set yrange [-3: 5] plot "lennard_↩ jones.dat" u 1:2 with line lt2 title "U (Lennard Jones' potential)

Notes: F (interaction based on Lennard Jones' potential) is repulsive when 0 <= r < 1.12 and attractive when r > 1.12 when r = 1.12, the 2 particles are in equilibrium (F=0).

Definition at line 22 of file test_lennard_jones.cpp.

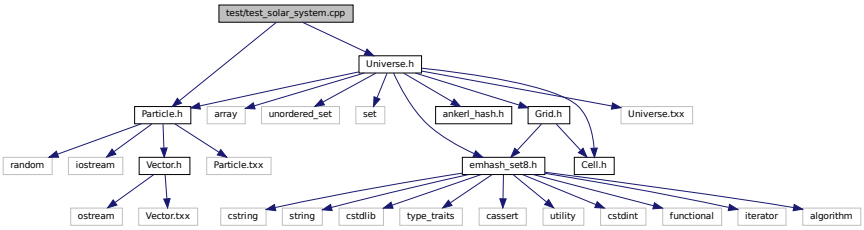
Here is the caller graph for this function:



7.23 test/test_solar_system.cpp File Reference

```
#include "Particle.h"
#include "Universe.h"
```

Include dependency graph for test_solar_system.cpp:



Functions

- int main ()

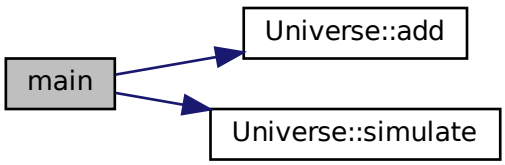
7.23.1 Function Documentation

7.23.1.1 main()

```
int main ( )
```

Definition at line 4 of file test_solar_system.cpp.

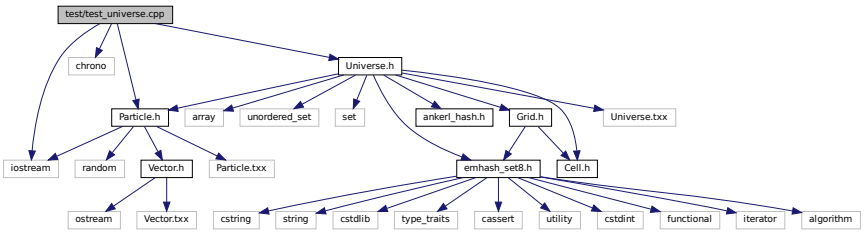
Here is the call graph for this function:



7.24 test/test_universe.cpp File Reference

```
#include <iostream>
#include <chrono>
#include "Particle.h"
#include "Universe.h"
```

Include dependency graph for test_universe.cpp:



Functions

- int main ()

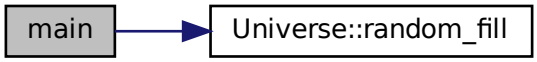
7.24.1 Function Documentation

7.24.1.1 main()

```
int main ( )
```

Definition at line 6 of file test_universe.cpp.

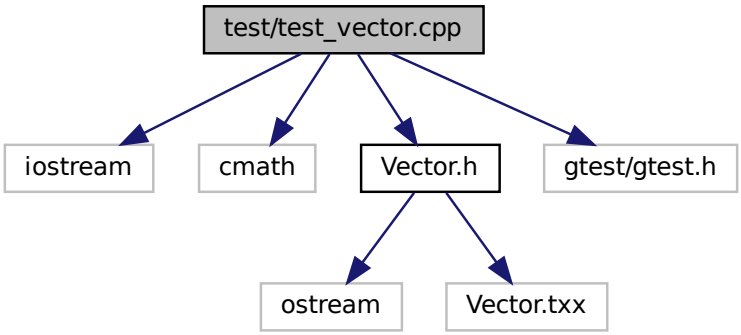
Here is the call graph for this function:



7.25 test/test_vector.cpp File Reference

```
#include <iostream>
#include <cmath>
#include "Vector.h"
#include <gtest/gtest.h>
```

Include dependency graph for test_vector.cpp:



Functions

- TEST (TestVector2, TestConstructors)
- TEST (TestVector2, TestMethods)
- TEST (TestVector2, TestOperators)
- TEST (TestVector3, TestConstructors)
- TEST (TestVector3, TestOperators)
- TEST (Testvector3, TestMethods)
- int main (int argc, char **argv)

7.25.1 Function Documentation

7.25.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

Definition at line 224 of file test_vector.cpp.

7.25.1.2 TEST() [1/6]

```
TEST (
            TestVector2 ,
            TestConstructors  )
```

Handles the tests for the Vector Class. The other classes (Universe, Particle, Grid) are tested along the different labs.

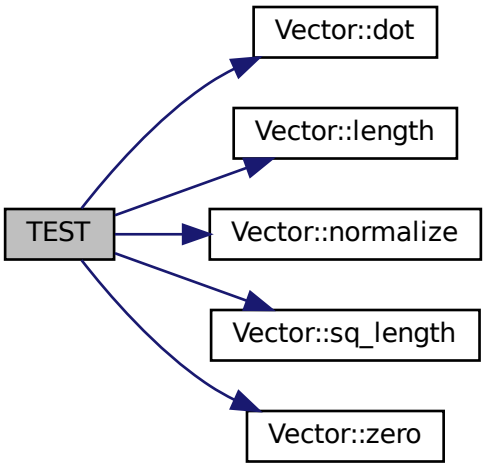
Definition at line 14 of file test_vector.cpp.

7.25.1.3 TEST() [2/6]

```
TEST (
            TestVector2 ,
            TestMethods  )
```

Definition at line 31 of file test_vector.cpp.

Here is the call graph for this function:



7.25.1.4 TEST() [3/6]

```
TEST (
            TestVector2 ,
            TestOperators  )
```

Definition at line 62 of file test_vector.cpp.

7.25.1.5 TEST() [4/6]

```
TEST (
            TestVector3 ,
            TestConstructors  )
```

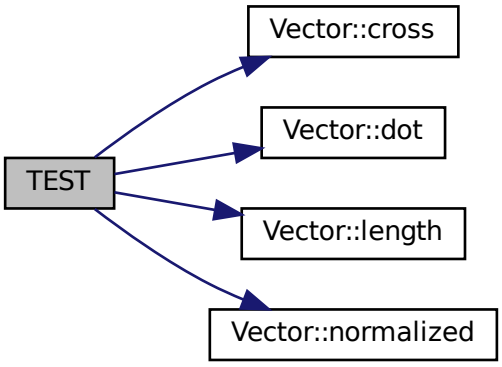
Definition at line 133 of file test_vector.cpp.

7.25.1.6 TEST() [5/6]

```
TEST (
            Testvector3 ,
            TestMethods  )
```

Definition at line 198 of file test_vector.cpp.

Here is the call graph for this function:



7.25.1.7 TEST() [6/6]

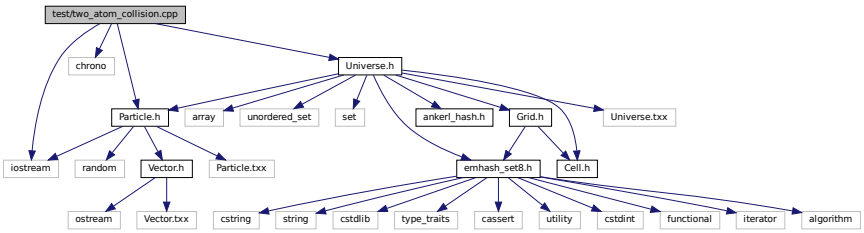
```
TEST (
            TestVector3 ,
            TestOperators  )
```

Definition at line 152 of file test_vector.cpp.

7.26 test/two_atom_collision.cpp File Reference

```
#include <iostream>
#include <chrono>
#include "Particle.h"
#include "Universe.h"
```

Include dependency graph for two_atom_collision.cpp:



Functions

- int main ()

7.26.1 Function Documentation

7.26.1.1 main()

```
int main ( )
```

Definition at line 9 of file two_atom_collision.cpp.

Here is the call graph for this function: