C++ AutoGarbage

0.1

Generated by Doxygen 1.8.19

1 Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

2 Hierarchical Index

2.1 Class Hierarchy

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

3 Class Index

3.1 Class List

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

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Namespace Documentation

5.1 gc Namespace Reference

Classes

- class array
- class **dyn_array_impl**
- class field
- class field< gc::array< T > >
- class **field< gc::dyn_array_impl< T > >**
- class object
- class static_ptr
- class static_ptr< gc::array< T > >
- class **static_ptr< gc::dyn_array_impl< T > >**

Functions

- void init (size_t heap_size, unsigned int max_allocation_attempts_before_gc)
- void debug ()
- void info ()

5.1.1 Function Documentation

5.1.1.1 debug() `void gc::debug ( )`

Print verbose debug information to the console about the current state of the AutoGarbage system.

5.1.1.2 info() `void gc::info ( )`

Print information to the console about the current state of the AutoGarbage system.

5.1.1.3 init()

```
void gc::init (
        size_t heap_size,
        unsigned int max_allocation_attempts_before_gc )
```

Initiates the AutoGarbage library. This must be called before attempting to allocate any gc::objects.

Parameters

heap_size	The size in bytes of the heap.
max_allocation_attempts_before_gc	The maximum number of times an allocation attempt is made for an object before the system gives up and collects garbage.

6 Class Documentation

6.1 gc::array< T > Class Template Reference

`#include <gc_array.h>`

Inheritance diagram for gc::array< T >:

Collaboration diagram for gc::array< T >:

Public Member Functions

- array (size_t array_size)
- gc::field< T > & operator[ ] (int i)

6.1.1 Detailed Description

template<class T>
class gc::array< T >

Creates an array of objects of the chosen type.

Parameters

<T>	The type of the elements of the array. This must a type of gc::object.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 array() template<class T >

```
gc::array< T >::array (
            size_t array_size )  [inline]
```

Creates an array of given size.

Parameters

array_size	Number of elements the array will be sized to. Arrays cannot be resized after their creation.

6.1.3 Member Function Documentation

6.1.3.1 operator[]() template<class T >

```
gc::field<T>& gc::array< T >::operator[] (
            int i )  [inline]
```

Accesses the elements of the array.

Parameters

i	Index of the array element to return.

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

- gclib/include/gc_array.h

6.2 gc::field< T > Class Template Reference

```
#include <gc_field.h>
```

Public Member Functions

- field ()
- field (T *object)

- field (const field< T > &f2)
- field< T > & operator= (const field< T > &f2)
- field< T > & operator= (T *o2)
- operator T* () const
- T * operator-> ()

6.2.1 Detailed Description

template<class T>
class gc::field< T >

Fields are used as garbage collectable members on inheritors of `gc::object`. Fields must appear as the first members of the object in order for the marker to find them. After all fields have been defined they must be suffixed with the `END_GC_FIELDS;` macro;

Parameters

<T>	The type of `gc::object` which is going to be referenced.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 field() [1/3] template<class T >

```
gc::field< T >::field ( )  [inline]
```

New field with null reference.

6.2.2.2 field() [2/3] template<class T >

```
gc::field< T >::field (
            T * object )  [inline]
```

New field pointing to given object.

Parameters

object	Object to reference.

6.2.2.3 field() [3/3] template<class T >

```
gc::field< T >::field (
            const field< T > & f2 )  [inline]
```

Copy constructor.

6.2.3 Member Function Documentation

6.2.3.1 operator T*() template<class T >

```
gc::field< T >::operator T* ( ) const  [inline]
```

Field will implicitly cast to the referenced array.

6.2.3.2 operator->() template<class T >

```
T* gc::field< T >::operator-> ( )  [inline]
```

Accesses members of the referenced object.

6.2.3.3 operator=() [1/2] template<class T >

```
field<T>& gc::field< T >::operator= (
            const field< T > & f2 )  [inline]
```

Copy assignment.

6.2.3.4 operator=() [2/2] template<class T >

```
field<T>& gc::field< T >::operator= (
            T * o2 )  [inline]
```

Assigns the field to the newly given object.

Parameters

object	New object to reference.

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

- gclib/include/gc_field.h

6.3 gc::field< gc::array< T > > Class Template Reference

```
#include <gc_field.h>
```

Public Member Functions

- field ()
- field (gc::array< T > *array_object)
- field (const field< T > &f2)
- field< gc::array< T > > & operator= (const field< gc::array< T >> &f2)
- field< gc::array< T > > & operator= (gc::array< T > *object)
- operator gc::array< T > * () const
- gc::field< T > & operator[ ] (int i)

6.3.1 Detailed Description

template<class T>
class gc::field< gc::array< T > >

Fields are used as garbage collectable members on inheritors of `gc::object`. Fields must appear as the first members of the object in order for the marker to find them. After all fields have been defined they must be suffixed with the `END_GC_FIELDS;` macro;

Parameters

<T>	The type of `gc::object` which is going to be used as elements to the referenced array.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 field() [1/3] template<class T >

```
gc::field< gc::array< T > >::field ( )  [inline]
```

New field with null reference.

6.3.2.2 field() [2/3] template<class T >

```
gc::field< gc::array< T > >::field (
            gc::array< T > * array_object )  [inline]
```

New field pointing to given array object.

Parameters

array_object	Array object to reference.

6.3.2.3 field() [3/3] template<class T >

```
gc::field< gc::array< T > >::field (
            const field< T > & f2 )  [inline]
```

Copy constructor.

6.3.3 Member Function Documentation

6.3.3.1 operator gc::array< T > *() template<class T >

```
gc::field< gc::array< T > >::operator gc::array< T > * ( ) const  [inline]
```

field will implicitly cast to the referenced array.

6.3.3.2 operator=() [1/2] template<class T >

```
field<gc::array<T> >& gc::field< gc::array< T > >::operator= (
            const field< gc::array< T >> & f2 ) [inline]
```

Copy assignment. Object is not recreated.

6.3.3.3 operator=() [2/2] template<class T >

```
field<gc::array<T> >& gc::field< gc::array< T > >::operator= (
            gc::array< T > * object ) [inline]
```

Assigns the field to the newly given array object.

Parameters

object	New array object to reference.

6.3.3.4 operator[]() template<class T >

```
gc::field<T>& gc::field< gc::array< T > >::operator[] (
            int i ) [inline]
```

Accesses the elements of the referenced array.

Parameters

i	Index of the array elemtn to return.

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

- gclib/include/gc_field.h

6.4 gc::object Class Reference

`#include <gc_object.h>`

Inheritance diagram for gc::object:

Collaboration diagram for gc::object:

Public Member Functions

- object ()
- void * operator new (size_t size)

6.4.1 Detailed Description

Users should override this class in order to make their object collectable by the AutoGarbage system.

User defined objects are expected to have the pattern of garbage collectable fields at the top, followed by the `END_GC_FIELDS;` macro, followed by any remaining non garbage collectable members.

Be careful when using raw/smart pointer members on `gc::object` classes as the destructor never gets called by the AutoGarbage system.

Example class:

```
class Foo : public gc::object
{
    private:
        gc::field<Bar> _gcField;
        gc::field<Bar> _otherGcField;
        END_GC_FIELDS;
        int _nonGcMember;
    public:
        Foo(Bar f1):
           _gcField(f1),
           _otherGcField(new Bar()),
           _nonGcMember(2)
        {
        }
        void deReference()
        {
            _gcField = nullptr;
        }
}
```

6.4.2 Constructor & Destructor Documentation

6.4.2.1 object() `gc::object::object ( )`

6.4.3 Member Function Documentation

6.4.3.1 operator new() `void* gc::object::operator new ( size_t size )`

Allocates a new object within the AutoGarbage library's heap.

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

- gclib/include/gc_object.h

6.5 gc::static_ptr< T > Class Template Reference

`#include <gc_static_ptr.h>`

Public Member Functions

- static_ptr ()
- static_ptr (T *object)
- static_ptr (const gc::static_ptr< T > &sptr2)
- gc::static_ptr< T > & operator= (const gc::static_ptr< T > &sptr2)
- gc::static_ptr< T > & operator= (T *object)
- operator T* () const
- ~static_ptr ()
- T * operator-> ()

6.5.1 Detailed Description

template<class T>
class gc::static_ptr< T >

static_ptr is used to keep a scoped reference to a garbage collectable object. Once the static_ptr object goes out of scope, the reference to the garbage collectable object is removed and can be collected as long as it is not referenced anywhere else.

Parameters

<T>	The type of `gc::object` which is going to be referenced.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 static_ptr() [1/3] template<class T >

```
gc::static_ptr< T >::static_ptr ( )  [inline]
```

New static_ptr with null reference.

6.5.2.2 static_ptr() [2/3] template<class T >

```
gc::static_ptr< T >::static_ptr (
            T * object )  [inline]
```

Assigns the static_ptr to the newly given object.

Parameters

object	New object to reference

6.5.2.3 static_ptr() [3/3] template<class T >

```
gc::static_ptr< T >::static_ptr (
            const gc::static_ptr< T > & sptr2 )  [inline]
```

Copy constructor.

6.5.2.4 ~static_ptr() template<class T >

gc::static_ptr< T >::~static_ptr ( ) [inline]

Once unscoped, the held object is now free to be collected (if it is not being referenced somewhere else).

6.5.3 Member Function Documentation

6.5.3.1 operator T*() template<class T >

gc::static_ptr< T >::operator T* ( ) const [inline]

static_ptr will implicitly cast to the referenced object.

6.5.3.2 operator->() template<class T >

T* gc::static_ptr< T >::operator-> ( ) [inline]

Accesses members of the referenced object.

6.5.3.3 operator=() [1/2] template<class T >

gc::static_ptr<T>& gc::static_ptr< T >::operator= (
const gc::static_ptr< T > & *sptr2*) [inline]

Copy assignment. Object is not recreated.

6.5.3.4 operator=() [2/2] template<class T >

gc::static_ptr<T>& gc::static_ptr< T >::operator= (
T * *object*) [inline]

Assigns the static_ptr to the newly given object.

Parameters

object	New object to reference.

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

- gclib/include/gc_object.h
- gclib/include/gc_static_ptr.h

6.6 gc::static_ptr< gc::array< T > > Class Template Reference

#include <gc_static_ptr.h>

Public Member Functions

- static_ptr ()
- static_ptr (gc::array< T > *array_object)
- static_ptr (const gc::static_ptr< gc::array< T >> &sptr2)
- gc::static_ptr< gc::array< T > > & operator= (const gc::static_ptr< gc::array< T >> &sptr2)
- gc::static_ptr< gc::array< T > > & operator= (gc::array< T > *object)
- operator gc::array< T > * () const
- gc::field< T > & operator[ ] (int i)
- ~static_ptr ()

6.6.1 Detailed Description

template<class T>
class gc::static_ptr< gc::array< T > >

static_ptr is used to keep a scoped reference to a garbage collectable object. Once the static_ptr object goes out of scope, the reference to the garbage collectable object is removed and can be collected as long as it is not referenced anywhere else.

Parameters

<T>	The type of `gc::object` which is going to be used as elements to the referenced array.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 static_ptr() [1/3] template<class T >

```
gc::static_ptr< gc::array< T > >::static_ptr ( )  [inline]
```

New static_ptr with null reference.

6.6.2.2 static_ptr() [2/3] template<class T >

```
gc::static_ptr< gc::array< T > >::static_ptr (
            gc::array< T > * array_object )  [inline]
```

Assigns the static_ptr to the newly given array object.

Parameters

object	New array object to reference

6.6.2.3 static_ptr() [3/3] template<class T >

```
gc::static_ptr< gc::array< T > >::static_ptr (
            const gc::static_ptr< gc::array< T >> & sptr2 )  [inline]
```

Copy constructor.

6.6.2.4 ~static_ptr() template<class T >

```
gc::static_ptr< gc::array< T > >::~static_ptr ( )  [inline]
```

Once unscoped, the held object is now free to be collected (if it is not being referenced somewhere else).

6.6.3 Member Function Documentation

6.6.3.1 operator gc::array< T > *() template<class T >

```
gc::static_ptr< gc::array< T > >::operator gc::array< T > * ( ) const  [inline]
```

static_ptr will implicitly cast to the referenced array.

6.6.3.2 operator=() [1/2] template<class T >

```
gc::static_ptr<gc::array<T> >& gc::static_ptr< gc::array< T > >::operator= (
            const gc::static_ptr< gc::array< T >> & sptr2 )  [inline]
```

Copy assignment. Object is not recreated.

6.6.3.3 operator=() [2/2] template<class T >

```
gc::static_ptr<gc::array<T> >& gc::static_ptr< gc::array< T > >::operator= (
            gc::array< T > * object )  [inline]
```

Assigns the static_ptr to the newly given array object.

Parameters

object	New array object to reference.

6.6.3.4 operator[]() template<class T >

```
gc::field<T>& gc::static_ptr< gc::array< T > >::operator[] (
            int i )  [inline]
```

Accesses the elements of the referenced array.

Parameters

i	Index of the array to return.

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

- gclib/include/gc_static_ptr.h

7 File Documentation

7.1 gclib/include/gc.h File Reference

```
#include "gc_array.h"
#include "gc_statics.h"
```
Include dependency graph for gc.h:

7.2 gclib/include/gc_array.h File Reference

```
#include "gc_heap.h"
#include "gc_dyn_array_impl.h"
#include "gc_object.h"
```
Include dependency graph for gc_array.h: This graph shows which files directly or indirectly include this file:

Classes

- class gc::array< T >

Namespaces

- gc

7.3 gclib/include/gc_field.h File Reference

```
#include "gc_heap.h"
```
Include dependency graph for gc_field.h: This graph shows which files directly or indirectly include this file:

Classes

- class gc::field< gc::array< T > >
- class gc::field< T >

Namespaces

- gc

7.4 gclib/include/gc_object.h File Reference

```
#include <iostream>
#include "gc_statics.h"
#include "gc_cell.h"
```
Include dependency graph for gc_object.h: This graph shows which files directly or indirectly include this file:

Classes

- class gc::object

Namespaces

- gc

Macros

- #define END_GC_FIELDS void* _end_gc_fields_magic_ptr = gc::heap::heap_struct::get()->end_gc_fields↩ _magic_ptr(); bool* gc_fields_end() override { return ((bool*)&_end_gc_fields_magic_ptr); };

7.4.1 Macro Definition Documentation

7.4.1.1 END_GC_FIELDS `#define END_GC_FIELDS void* _end_gc_fields_magic_ptr = gc::heap↩ ::heap_struct::get()->end_gc_fields_magic_ptr(); bool* gc_fields_end() override { return ((bool*)&↩ _end_gc_fields_magic_ptr); };`

This macro must be placed on inheritors of objects to signify the the marking system that there are no more fields left on the object.

7.5 gclib/include/gc_static_ptr.h File Reference

`#include "gc_heap.h"`
Include dependency graph for gc_static_ptr.h: This graph shows which files directly or indirectly include this file:

Classes

- class gc::static_ptr< gc::array< T > >
- class gc::static_ptr< T >

Namespaces

- gc

7.6 gclib/include/gc_statics.h File Reference

`#include <cstdlib>`
Include dependency graph for gc_statics.h: This graph shows which files directly or indirectly include this file:

Namespaces

- gc

Functions

- void gc::init (size_t heap_size, unsigned int max_allocation_attempts_before_gc)
- void gc::debug ()
- void gc::info ()

Index