My Project

Generated by Doxygen 1.8.14

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

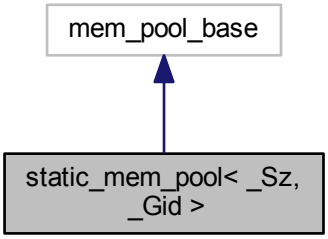
Chapter 4

Class Documentation

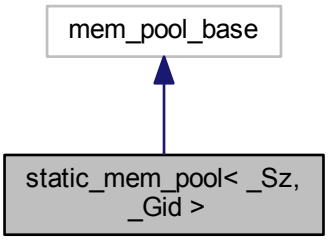
4.1 static_mem_pool< _Sz, _Gid > Class Template Reference

`#include <static_mem_pool.h>`

Inheritance diagram for static_mem_pool< _Sz, _Gid >:



Collaboration diagram for static_mem_pool< _Sz, _Gid >:



Public Member Functions

- void * allocate ()
- void deallocate (void *__ptr)
- virtual void recycle ()

Static Public Member Functions

- static static_mem_pool & instance ()
- static static_mem_pool & instance_known ()

4.1.1 Detailed Description

template<size_t _Sz, int _Gid = -1>
class static_mem_pool< _Sz, _Gid >

Singleton class template to manage the allocation/deallocation of memory blocks of one specific size.

Parameters

_Sz	size of elements in the static_mem_pool
_Gid	group id of a static_mem_pool: if it is negative, simultaneous accesses to this static_mem_pool will be protected from each other; otherwise no protection is given

4.1.2 Member Function Documentation

4.1.2.1 allocate()

```
template<size_t _Sz, int _Gid = -1>
void* static_mem_pool< _Sz, _Gid >::allocate ( )  [inline]
```

Allocates memory and returns its pointer. The template will try to get it from the memory pool first, and request memory from the system if there is no free memory in the pool.

Returns

pointer to allocated memory if successful; NULL otherwise

4.1.2.2 deallocate()

```
template<size_t _Sz, int _Gid = -1>
void static_mem_pool< _Sz, _Gid >::deallocate (
            void * __ptr )  [inline]
```

Deallocates memory by putting the memory block into the pool.

Parameters

__ptr	pointer to memory to be deallocated

4.1.2.3 instance()

```
template<size_t _Sz, int _Gid = -1>
static static_mem_pool& static_mem_pool< _Sz, _Gid >::instance ( )  [inline], [static]
```

Gets the instance of the static memory pool. It will create the instance if it does not already exist. Generally this function is now not needed.

Returns

reference to the instance of the static memory pool

See also

instance_known

4.1.2.4 instance_known()

```
template<size_t _Sz, int _Gid = -1>
static static_mem_pool& static_mem_pool< _Sz, _Gid >::instance_known ( )  [inline], [static]
```

Gets the known instance of the static memory pool. The instance must already exist. Generally the static initializer of the template guarantees it.

Returns

reference to the instance of the static memory pool

4.1.2.5 recycle()

```
template<size_t _Sz, int _Gid>
void static_mem_pool< _Sz, _Gid >::recycle ( )  [virtual]
```

Recycles half of the free memory blocks in the memory pool to the system. It is called when a memory request to the system (in other instances of the static memory pool) fails.

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

- static_mem_pool.h

4.2 static_mem_pool_set Class Reference

`#include <static_mem_pool.h>`

Public Types

- typedef class_level_lock< static_mem_pool_set >::lock **lock**

Public Member Functions

- void **recycle** ()
- void **add** (mem_pool_base *__memory_pool_p)

Static Public Member Functions

- static static_mem_pool_set & **instance** ()

Public Attributes

- __PRIVATE **__pad0__**: ~static_mem_pool_set()

4.2.1 Detailed Description

Singleton class to maintain a set of existing instantiations of static_mem_pool.

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

- static_mem_pool.h

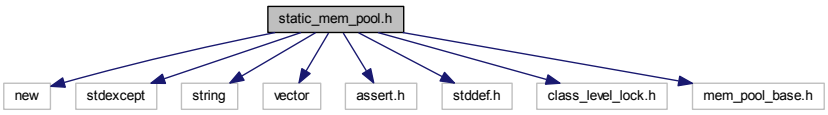
Chapter 5

File Documentation

5.1 static_mem_pool.h File Reference

```
#include <new>
#include <stdexcept>
#include <string>
#include <vector>
#include <assert.h>
#include <stddef.h>
#include "class_level_lock.h"
#include "mem_pool_base.h"
```

Include dependency graph for static_mem_pool.h:



Classes

- class static_mem_pool_set
- class static_mem_pool< _Sz, _Gid >

Macros

- #define **__PRIVATE** private
- #define **_STATIC_MEM_POOL_TRACE**(_Lck, _Msg) ((void)0)
- #define **DECLARE_STATIC_MEM_POOL**(_Cls)
- #define **DECLARE_STATIC_MEM_POOL__NOTHROW**(_Cls)
- #define **DECLARE_STATIC_MEM_POOL_GROUPED**(_Cls, _Gid)
- #define **DECLARE_STATIC_MEM_POOL_GROUPED__NOTHROW**(_Cls, _Gid)
- #define **PREPARE_STATIC_MEM_POOL**(_Cls) std::cerr << "PREPARE_STATIC_MEM_POOL is obsolete!\n";
- #define **PREPARE_STATIC_MEM_POOL_GROUPED**(_Cls, _Gid) std::cerr << "PREPARE_STATIC_M↩EM_POOL_GROUPED is obsolete!\n";

5.1.1 Detailed Description

Header file for the 'static' memory pool.

Version

1.20, 2007/10/20

Author

Wu Yongwei

5.1.2 Macro Definition Documentation

5.1.2.1 DECLARE_STATIC_MEM_POOL

```
#define DECLARE_STATIC_MEM_POOL(
              _Cls )
```

Value:

```
public: \
    static void* operator new(size_t __size) \
    { \
        assert(__size == sizeof(_Cls)); \
        void* __ptr; \
        __ptr = static_mem_pool<sizeof(_Cls)>:: \
                                instance_known().allocate(); \
        if (__ptr == NULL) \
            throw std::bad_alloc(); \
        return __ptr; \
    } \
    static void operator delete(void* __ptr) \
    { \
        if (__ptr) \
            static_mem_pool<sizeof(_Cls)>:: \
                        instance_known().deallocate(__ptr); \
    }
```

5.1.2.2 DECLARE_STATIC_MEM_POOL__NOTHROW

```
#define DECLARE_STATIC_MEM_POOL__NOTHROW(
              _Cls )
```

Value:

```
public: \
    static void* operator new(size_t __size) throw() \
    { \
        assert(__size == sizeof(_Cls)); \
        return static_mem_pool<sizeof(_Cls)>:: \
                            instance_known().allocate(); \
    } \
    static void operator delete(void* __ptr) \
    { \
        if (__ptr) \
            static_mem_pool<sizeof(_Cls)>:: \
                        instance_known().deallocate(__ptr); \
    }
```

5.1.2.3 DECLARE_STATIC_MEM_POOL_GROUPED

```
#define DECLARE_STATIC_MEM_POOL_GROUPED(
                _Cls,
                _Gid )
```

Value:

```
public: \
    static void* operator new(size_t __size) \
    { \
        assert(__size == sizeof(_Cls)); \
        void* __ptr; \
        __ptr = static_mem_pool<sizeof(_Cls), (_Gid)>:: \
                                instance_known().allocate(); \
        if (__ptr == NULL) \
            throw std::bad_alloc(); \
        return __ptr; \
    } \
    static void operator delete(void* __ptr) \
    { \
        if (__ptr) \
            static_mem_pool<sizeof(_Cls), (_Gid)>:: \
                            instance_known().deallocate(__ptr); \
    }
```

5.1.2.4 DECLARE_STATIC_MEM_POOL_GROUPED__NOTHROW

```
#define DECLARE_STATIC_MEM_POOL_GROUPED__NOTHROW(
                _Cls,
                _Gid )
```

Value:

```
public: \
    static void* operator new(size_t __size) throw() \
    { \
        assert(__size == sizeof(_Cls)); \
        return static_mem_pool<sizeof(_Cls), (_Gid)>:: \
                               instance_known().allocate(); \
    } \
    static void operator delete(void* __ptr) \
    { \
        if (__ptr) \
            static_mem_pool<sizeof(_Cls), (_Gid)>:: \
                            instance_known().deallocate(__ptr); \
    }
```

Index