

libdebug

Generated by Doxygen 1.9.4

1 Todo List	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 MemBlock Struct Reference	7
4.1.1 Detailed Description	8
4.1.2 Member Data Documentation	8
4.1.2.1 CompilDate	8
4.1.2.2 CompilTime	8
4.1.2.3 File	8
4.1.2.4 Function	8
4.1.2.5 Line	9
4.1.2.6 Next	9
4.1.2.7 Prev	9
4.1.2.8 Ptr	9
4.1.2.9 Size	9
5 File Documentation	11
5.1 assert.h File Reference	11
5.1.1 Detailed Description	12
5.1.2 Macro Definition Documentation	12
5.1.2.1 ASSERT	13
5.1.2.2 DBG_ITRACE	13
5.1.2.3 DBG_MSG	13
5.1.2.4 DBG_PRINTF	14
5.1.2.5 DBG_TRACE	14
5.1.3 Function Documentation	14
5.1.3.1 _trace()	14
5.1.3.2 _trace_dynmsg()	15
5.1.3.3 _trace_msg()	15
5.2 assert.h	16
5.3 memdbg.h File Reference	17
5.3.1 Detailed Description	18
5.3.2 Function Documentation	19
5.3.2.1 dbg_asprintf()	19
5.3.2.2 dbg_calloc()	19
5.3.2.3 dbg_free()	20
5.3.2.4 dbg_malloc()	21
5.3.2.5 dbg_realloc()	22

5.3.2.6 dbg_strdup()	24
5.4 memdbg.h	24
5.5 memory.h File Reference	25
5.5.1 Detailed Description	26
5.5.2 Macro Definition Documentation	27
5.5.2.1 asprintf	27
5.5.2.2 calloc	27
5.5.2.3 free	27
5.5.2.4 malloc	28
5.5.2.5 memreport	28
5.5.2.6 realloc	28
5.5.2.7 strdup	28
5.6 memory.h	29
5.7 memtrack.h File Reference	29
5.7.1 Detailed Description	31
5.7.2 Typedef Documentation	31
5.7.2.1 TMemBlock	31
5.7.3 Variable Documentation	31
5.7.3.1 memtrack_addblock	31
5.7.3.2 memtrack_delblock	32
5.7.3.3 memtrack_dumpblocks	33
5.7.3.4 memtrack_getallocatedblocks	33
5.7.3.5 memtrack_getallocatedRAM	34
5.7.3.6 memtrack_getblocksize	34
5.8 memtrack.h	34
5.9 assert.c File Reference	35
5.9.1 Detailed Description	36
5.9.2 Macro Definition Documentation	36
5.9.2.1 _XOPEN_SOURCE	36
5.9.3 Function Documentation	37
5.9.3.1 _trace()	37
5.9.3.2 _trace_dynmsg()	37
5.9.3.3 _trace_msg()	38
5.10 assert.c	38
5.11 memdbg.c File Reference	40
5.11.1 Detailed Description	41
5.11.2 Macro Definition Documentation	41
5.11.2.1 _GNU_SOURCE	41
5.11.2.2 _XOPEN_SOURCE	41
5.11.3 Function Documentation	41
5.11.3.1 dbg_asprintf()	41
5.11.3.2 dbg_calloc()	42

5.11.3.3 <code>dbg_free()</code>	43
5.11.3.4 <code>dbg_malloc()</code>	44
5.11.3.5 <code>dbg_realloc()</code>	45
5.11.3.6 <code>dbg_strdup()</code>	46
5.12 <code>memdbg.c</code>	47
5.13 <code>memtrack.c</code> File Reference	49
5.13.1 Detailed Description	51
5.13.2 Macro Definition Documentation	51
5.13.2.1 <code>_XOPEN_SOURCE</code>	51
5.13.3 Function Documentation	51
5.13.3.1 <code>memtrack_reset()</code>	51
5.13.4 Variable Documentation	52
5.13.4.1 <code>memtrack_addblock</code>	52
5.13.4.2 <code>memtrack_delblock</code>	53
5.13.4.3 <code>memtrack_dumpblocks</code>	54
5.13.4.4 <code>memtrack_getallocatedblocks</code>	54
5.13.4.5 <code>memtrack_getallocatedRAM</code>	55
5.13.4.6 <code>memtrack_getblocksize</code>	55
5.14 <code>memtrack.c</code>	55
5.15 <code>revision.h</code> File Reference	61
5.15.1 Macro Definition Documentation	61
5.15.1.1 <code>REVISION</code>	61
5.16 <code>revision.h</code>	61
Index	63

Chapter 1

Todo List

Member `_trace_dynmsg` (const char *p_File, const unsigned int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function, const char *p_Format,...)

Replace with a portable snprintf function

Member `dbg_asprintf` (char **p_Ptr, const char *p_Format, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function,...)

Implement a vasprintf wrapping function to catch allocation and use it here

Chapter 2

Class Index

2.1 Class List

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

MemBlock	
Memory block metadata list item	7

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

assert.h	Debugging macros	11
memdbg.h	Memory leak tracker header	17
memory.h	Tracks memory allocation and leaks when compiled without NDEBUG	25
memtrack.h	Memory block metadata tracking headers	29
assert.c	Compiled functions used by debugging macros to write on stderr	35
memdbg.c	Memory leak tracker implementation	40
memtrack.c	Memory block metadata tracking implementation	49
revision.h		61

Chapter 4

Class Documentation

4.1 MemBlock Struct Reference

Memory block metadata list item.

```
#include <memtrack.h>
```

Collaboration diagram for MemBlock:



Public Attributes

- struct [MemBlock](#) * [Prev](#)
Previous item pointer.
- struct [MemBlock](#) * [Next](#)
Next item pointer.
- void * [Ptr](#)
Allocated memory block pointer.
- size_t [Size](#)
Allocated memory block size.
- char * [File](#)
Source file which asked the allocation.
- int [Line](#)
Source line number ch asked the allocation.
- char * [CompilDate](#)
Source file compilation date.
- char * [CompilTime](#)
Source file compilation time.
- char * [Function](#)
Fonction name which asked the allocation.

4.1.1 Detailed Description

Memory block metadata list item.

Double linked list item to store memory block metadata

Definition at line 33 of file [memtrack.h](#).

4.1.2 Member Data Documentation

4.1.2.1 CompilDate

```
char* MemBlock::CompilDate
```

Source file compilation date.

Definition at line 40 of file [memtrack.h](#).

4.1.2.2 CompilTime

```
char* MemBlock::CompilTime
```

Source file compilation time.

Definition at line 41 of file [memtrack.h](#).

4.1.2.3 File

```
char* MemBlock::File
```

Source file which asked the allocation.

Definition at line 38 of file [memtrack.h](#).

4.1.2.4 Function

```
char* MemBlock::Function
```

Fonction name which asked the allocation.

Definition at line 42 of file [memtrack.h](#).

4.1.2.5 Line

```
int MemBlock::Line
```

Source line number ch asked the allocation.

Definition at line 39 of file [memtrack.h](#).

4.1.2.6 Next

```
struct MemBlock* MemBlock::Next
```

Next item pointer.

Definition at line 35 of file [memtrack.h](#).

4.1.2.7 Prev

```
struct MemBlock* MemBlock::Prev
```

Previous item pointer.

Definition at line 34 of file [memtrack.h](#).

4.1.2.8 Ptr

```
void* MemBlock::Ptr
```

Allocated memory block pointer.

Definition at line 36 of file [memtrack.h](#).

4.1.2.9 Size

```
size_t MemBlock::Size
```

Allocated memory block size.

Definition at line 37 of file [memtrack.h](#).

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

- [memtrack.h](#)

Chapter 5

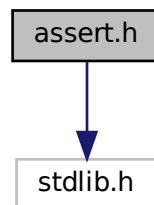
File Documentation

5.1 assert.h File Reference

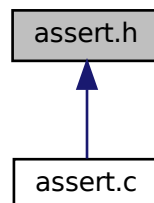
Debugging macros.

```
#include <stdlib.h>
```

Include dependency graph for assert.h:



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



Macros

- #define `ASSERT`(condition)
Assertion check macro.
- #define `DBG_TRACE _trace`(__FILE__, __LINE__, __DATE__, __TIME__, __func__)
Checkpoint on stderr.
- #define `DBG_MSG`(msg) `_trace_msg`(__FILE__, __LINE__, __DATE__, __TIME__, __func__, msg)
Checkpoint on stderr with a static message.
- #define `DBG_ITRACE`(inst)
Instruction checkpoint.
- #define `DBG_PRINTF`(p_Format, ...) `_trace_dynmsg`(__FILE__, __LINE__, __DATE__, __TIME__, __func__, p_Format, __VA_ARGS__)
Log a timestamped debugging message on stderr.

Functions

- void `_trace` (const char *p_File, const unsigned int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function)
Print a debug trace (checkpoint)
- void `_trace_msg` (const char *p_File, const unsigned int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function, const char *p_Message)
Print a debug trace (checkpoint) with a static message.
- void `_trace_dynmsg` (const char *p_File, const unsigned int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function, const char *p_Format,...)
Print a debug trace (checkpoint) with a formatted message.

5.1.1 Detailed Description

Debugging macros.

Date

11/05/1997

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 1997-2024, François Cerbelle

Originally inspired by "L'art du code", Steve Maguire, Microsoft Press

Definition in file [assert.h](#).

5.1.2 Macro Definition Documentation

5.1.2.1 ASSERT

```
#define ASSERT(  
    condition )
```

Value:

```
if (!(condition)) { \
    _trace_dynmsg(__FILE__, __LINE__, __DATE__, __TIME__, __func__, "Assertion failed (%s)" , #condition);  
    \
    abort(); \
}
```

Assertion check macro.

Parameters

in	<i>condition</i>	to check
----	------------------	----------

If NDEBUG is set, does nothing. If NDEBUG is not defined, checks that the condition is true, otherwise stop the process

Definition at line 104 of file [assert.h](#).

5.1.2.2 DBG_ITRACE

```
#define DBG_ITRACE(  
    inst )
```

Value:

```
_trace_msg(__FILE__, __LINE__, __DATE__, __TIME__, __func__, #inst), \
inst
```

Instruction checkpoint.

Writes a checkpoint trace with timestamp, filename, function name and line number when executing an instruction.

Definition at line 143 of file [assert.h](#).

5.1.2.3 DBG_MSG

```
#define DBG_MSG(  
    msg ) _trace_msg(__FILE__, __LINE__, __DATE__, __TIME__, __func__, msg)
```

Checkpoint on stderr with a static message.

Writes a timestamped checkpoint with filename, function and line number on stderr.

Definition at line 131 of file [assert.h](#).

5.1.2.4 DBG_PRINTF

```
#define DBG_PRINTF(
    p_Format,
    ... )    _trace_dynmsg(__FILE__, __LINE__, __DATE__, __TIME__, __func__, p_↵
Format, __VA_ARGS__)
```

Log a timestamped debugging message on stderr.

Writes a timestamped message on stderr with the filename, function name, line number.

Definition at line 156 of file [assert.h](#).

5.1.2.5 DBG_TRACE

```
#define DBG_TRACE    _trace(__FILE__, __LINE__, __DATE__, __TIME__, __func__)
```

Checkpoint on stderr.

Writes a timestamped checkpoint with filename, function and line number on stderr.

Definition at line 119 of file [assert.h](#).

5.1.3 Function Documentation

5.1.3.1 _trace()

```
void _trace (
    const char * p_File,
    const unsigned int p_Line,
    const char * p_CompilDate,
    const char * p_CompilTime,
    const char * p_Function )
```

Print a debug trace (checkpoint)

Parameters

in	<i>p_File</i>	Source file
in	<i>p_Line</i>	Source line in the source file
in	<i>p_CompilDate</i>	Compilation date
in	<i>p_CompilTime</i>	Compilation time
in	<i>p_Function</i>	Function name in the source file

Outputs on stderr a timestamp, with the filename, the sourceline, the compilation date and time, the function name.

Definition at line 77 of file [assert.c](#).

5.1.3.2 _trace_dynmsg()

```
void _trace_dynmsg (
    const char * p_File,
    const unsigned int p_Line,
    const char * p_CompilDate,
    const char * p_CompilTime,
    const char * p_Function,
    const char * p_Format,
    ... )
```

Print a debug trace (checkpoint) with a formatted message.

Parameters

in	<i>p_File</i>	Source file
in	<i>p_Line</i>	Source line in the source file
in	<i>p_CompilDate</i>	Compilation date
in	<i>p_CompilTime</i>	Compilation time
in	<i>p_Function</i>	Function name in the source file
in	<i>p_Format</i>	format string
in	...	Formatted string parameters

Outputs on stderr a timestamp, with the filename, the sourceline, the compilation date and time, the function name and a formatted message.

Todo Replace with a portable snprintf function

Definition at line 101 of file [assert.c](#).

5.1.3.3 _trace_msg()

```
void _trace_msg (
    const char * p_File,
    const unsigned int p_Line,
    const char * p_CompilDate,
    const char * p_CompilTime,
    const char * p_Function,
    const char * p_Message )
```

Print a debug trace (checkpoint) with a static message.

Parameters

in	<i>p_File</i>	Source file
----	---------------	-------------

Parameters

in	<i>p_Line</i>	Source line in the source file
in	<i>p_CompilDate</i>	Compilation date
in	<i>p_CompilTime</i>	Compilation time
in	<i>p_Function</i>	Function name in the source file
in	<i>p_Message</i>	Static message

Outputs on stderr a timestamp, with the filename, the sourceline, the compilation date and time, the function name and a static message.

Definition at line 88 of file [assert.c](#).

5.2 assert.h

[Go to the documentation of this file.](#)

```

00001
00019 #ifndef __ASSERT_H__
00020 #define __ASSERT_H__
00021
00022 #ifdef HAVE_CONFIG_H
00023 #include "config.h"
00024 #endif
00025
00026 #include <stdlib.h> /* abort */
00027
00028 #ifndef NDEBUG
00029
00030 #ifdef __cplusplus
00031 extern "C" {
00032 #endif
00033
00045 void _trace(const char *p_File,
00046            const unsigned int p_Line,
00047            const char *p_CompilDate,
00048            const char *p_CompilTime, const char *p_Function);
00049
00063 void _trace_msg(const char *p_File,
00064                const unsigned int p_Line,
00065                const char *p_CompilDate,
00066                const char *p_CompilTime,
00067                const char *p_Function, const char *p_Message);
00068
00083 void _trace_dynmsg(const char *p_File,
00084                   const unsigned int p_Line,
00085                   const char *p_CompilDate,
00086                   const char *p_CompilTime,
00087                   const char *p_Function, const char *p_Format, ...
00088                   );
00089
00090 #ifdef __cplusplus
00091 }
00092 #endif
00093
00094 #    endif                /* NDEBUG */
00095
00103 #ifndef NDEBUG
00104 #define ASSERT(condition) \
00105 if (!(condition)) { \
00106 _trace_dynmsg(__FILE__, __LINE__, __DATE__, __TIME__, __func__, "Assertion failed (%s)" , #condition); \
00107 abort(); \
00108 }
00109 #else
00110 #define ASSERT(condition) /* NDEBUG */
00111 #endif
00112 /* NDEBUG */
00118 #ifndef NDEBUG
00119 #define DBG_TRACE \
00120 _trace(__FILE__, __LINE__, __DATE__, __TIME__, __func__)
00121 #else
00122 #define DBG_TRACE /* NDEBUG */
00123 #endif
00124 /* NDEBUG */

```

```

00124
00130 #ifndef NDEBUG
00131 #define DBG_MSG(msg) \
00132 _trace_msg(__FILE__, __LINE__, __DATE__, __TIME__, __func__, msg)
00133 #else
00134 #define DBG_MSG(msg)
00135 #endif
00136
00142 #ifndef NDEBUG
00143 #define DBG_ITRACE(inst) \
00144 _trace_msg(__FILE__, __LINE__, __DATE__, __TIME__, __func__, #inst), \
00145 inst
00146 #else
00147 #define DBG_ITRACE(inst) inst
00148 #endif
00149
00155 #ifndef NDEBUG
00156 #define DBG_PRINTF(p_Format, ...) \
00157 _trace_dynmsg(__FILE__, __LINE__, __DATE__, __TIME__, __func__, p_Format, __VA_ARGS__)
00158 #else
00159 #define DBG_PRINTF(p_Format, ...)
00160 #endif
00161
00162 #endif
00163
00164 /* vim: set tw=80: */

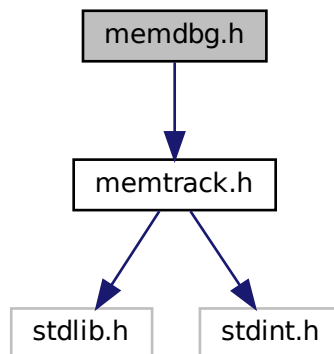
```

5.3 memdbg.h File Reference

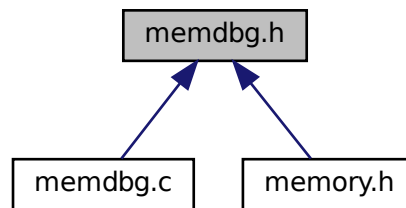
Memory leak tracker header.

```
#include "memtrack.h"
```

Include dependency graph for memdbg.h:



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



Functions

- void * [dbg_malloc](#) (const size_t Size, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function)
Malloc compatible standard allocation.
- void [dbg_free](#) (void *Ptr, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function)
Free compatible standard memory release.
- void * [dbg_calloc](#) (const size_t NMem, const size_t Size, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function)
Allocate a table of item from the size of each and number.
- void * [dbg_realloc](#) (void *Ptr, const size_t Size, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function)
Resize an already allocated and tracked block.
- char * [dbg_strdup](#) (const char *Ptr, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function)
String duplication with allocation.
- int [dbg_asprintf](#) (char **p_Ptr, const char *p_Format, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function,...)
Build a formatted string with allocation.

5.3.1 Detailed Description

Memory leak tracker header.

Date

25/09/2006

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 1997-2024, François Cerbelle

Originally inspired by "L'art du code", Steve Maguire, Microsoft Press

Definition in file [memdbg.h](#).

5.3.2 Function Documentation

5.3.2.1 dbg_asprintf()

```
int dbg_asprintf (
    char ** p_Ptr,
    const char * p_Format,
    const char * File,
    const int Line,
    const char * CompilDate,
    const char * CompilTime,
    const char * Function,
    ... )
```

Build a formatted string with allocation.

Parameters

in, out	<i>p_Ptr</i>	: Pointer on the to be built string
in	<i>p_Format</i>	: Pointer on the format string
in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompilDate</i>	: File compilation date
in	<i>CompilTime</i>	: File compilation time
in	<i>Function</i>	: Source function name
in	...	: Values referenced by the format string

Returns

Status of the formatting

Return values

≥ 0	number of chars in the output string (as strdup)
-1	in case of error (*p_Ptr contents is undefined)

Todo Implement a vasprintf wrapping function to catch allocation and use it here

Definition at line 200 of file [memdbg.c](#).

5.3.2.2 dbg_calloc()

```
void * dbg_calloc (
    const size_t NMem,
```

```

    const size_t Size,
    const char * File,
    const int Line,
    const char * CompilDate,
    const char * CompilTime,
    const char * Function )

```

Allocate a table of item from the size of each and number.

Uses malloc to allocate and track the memory bloc. If allocation and tracking succeed, fill the memory block with zeros.

Parameters

in	<i>NMem</i>	: Item number in the table
in	<i>Size</i>	: Item size in bytes
in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompilDate</i>	: File compilation date
in	<i>CompilTime</i>	: File compilation time
in	<i>Function</i>	: Source function name

Returns

Allocated block address

Return values

<i>Address</i>	if succeeded,
<i>NULL</i>	if not possible.

Definition at line 83 of file [memdbg.c](#).

Here is the call graph for this function:



5.3.2.3 dbg_free()

```

void dbg_free (
    void * Ptr,

```

```

const char * File,
const int Line,
const char * CompilDate,
const char * CompilTime,
const char * Function )

```

Free compatible standard memory release.

If the *Ptr* value is not NULL, try to untrack it. If it can not be found in the tracked list, because it was not tracked, it was allocated by a non monitored function, the function abort the process for investigation (missing a monitoring macro/function in the tracker, bug in the tracker, allocation from an external non-instrumented library). If the *Ptr* value was tracked, found and removed from the list successfully, forward it to free for actual free.

Parameters

in	<i>Ptr</i>	: Pointer on the memory to free
in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompilDate</i>	: File compilation date
in	<i>CompilTime</i>	: File compilation time
in	<i>Function</i>	: Source function name

Definition at line 60 of file [memdbg.c](#).

Here is the caller graph for this function:



5.3.2.4 dbg_malloc()

```

void * dbg_malloc (
    const size_t Size,
    const char * File,
    const int Line,
    const char * CompilDate,
    const char * CompilTime,
    const char * Function )

```

Malloc compatible standard allocation.

Parameters

in	<i>Size</i>	: Requested size in bytes
----	-------------	---------------------------

Parameters

in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompilDate</i>	: File compilation date
in	<i>CompilTime</i>	: File compilation time
in	<i>Function</i>	: Source function name

Returns

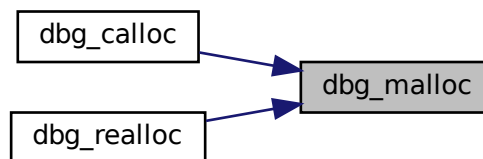
Allocated block address

Return values

<i>Address</i>	if succeeded,
<i>NULL</i>	if not possible.

Definition at line 27 of file [memdbg.c](#).

Here is the caller graph for this function:

**5.3.2.5 dbg_realloc()**

```
void * dbg_realloc (
    void * Ptr,
    const size_t Size,
    const char * File,
    const int Line,
    const char * CompilDate,
    const char * CompilTime,
    const char * Function )
```

Resize an already allocated and tracked block.

This function always uses malloc to allocate a new block and force an address change, and a data loss in case of shrink, which is the worst case scenario for realloc.

As for `realloc`, a `NULL` source pointer makes `realloc` act as `malloc`, and a new size set to 0 acts as `free`. The input pointer is left untouched but should not be used or freed anymore. It is always different than the return value. The input values referenced by the input pointer are not wiped.

This implementation can not be used if `realloc` is used to reduce a huge bloc in order to manage an OOM situation. Real `realloc` can succeed by actually downsizing the same memory block, inplace, but this implementation will fail because it first allocate a new block.

Parameters

in	<i>Ptr</i>	: Pointer on the memory to resize
in	<i>Size</i>	: New size in bytes
in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompilDate</i>	: File compilation date
in	<i>CompilTime</i>	: File compilation time
in	<i>Function</i>	: Source function name

Returns

Resized block address

Return values

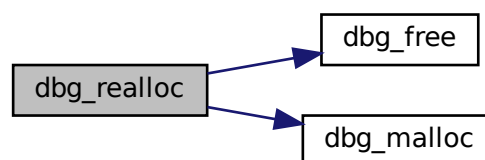
<i>Address</i>	if succeeded,
<i>NULL</i>	if not possible.

< Existing bloc size

< New block

Definition at line 108 of file [memdbg.c](#).

Here is the call graph for this function:



5.3.2.6 dbg_strdup()

```
char * dbg_strdup (
    const char * Ptr,
    const char * File,
    const int Line,
    const char * CompilDate,
    const char * CompilTime,
    const char * Function )
```

String duplication with allocation.

Parameters

in	<i>Ptr</i>	: Pointer on the string to copy
in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompilDate</i>	: File compilation date
in	<i>CompilTime</i>	: File compilation time
in	<i>Function</i>	: Source function name

Returns

Copied string address

Return values

<i>Address</i>	if succeeded,
<i>NULL</i>	if not possible.

Copy address

Definition at line 168 of file [memdbg.c](#).

5.4 memdbg.h

[Go to the documentation of this file.](#)

```
00001
00019 #ifndef __MEMDBG_H__
00020 #define __MEMDBG_H__
00021
00022 #ifdef HAVE_CONFIG_H
00023 # include "config.h"
00024 #endif
00025
00026 #include "memtrack.h"
00027
00028 #ifdef __cplusplus
00029 extern "C" {
00030 #endif
00031
00046 void *dbg_malloc(const size_t Size,
00047                 const char *File,
00048                 const int Line,
00049                 const char *CompilDate,
00050                 const char *CompilTime,
00051                 const char *Function);
```

```

00052
00071 void dbg_free(void *Ptr,
00072               const char *File,
00073               const int Line,
00074               const char *CompilDate,
00075               const char *CompilTime,
00076               const char *Function);
00077
00096 void *dbg_calloc(const size_t NMem,
00097                  const size_t Size,
00098                  const char *File,
00099                  const int Line,
00100                  const char *CompilDate,
00101                  const char *CompilTime,
00102                  const char *Function);
00103
00132 void *dbg_realloc(void *Ptr,
00133                   const size_t Size,
00134                   const char *File,
00135                   const int Line,
00136                   const char *CompilDate,
00137                   const char *CompilTime,
00138                   const char *Function);
00139
00154 char *dbg_strdup(const char *Ptr,
00155                  const char *File,
00156                  const int Line,
00157                  const char *CompilDate,
00158                  const char *CompilTime,
00159                  const char *Function);
00160
00177 int dbg_asprintf(char **p_Ptr,
00178                  const char *p_Format,
00179                  const char *File,
00180                  const int Line,
00181                  const char *CompilDate,
00182                  const char *CompilTime,
00183                  const char *Function,
00184                  ...);
00185
00186 #ifdef __cplusplus
00187 }
00188 #endif
00189
00190 #endif /* __MEMDBG_H__ */
00191
00192 /* vim: set tw=80: */

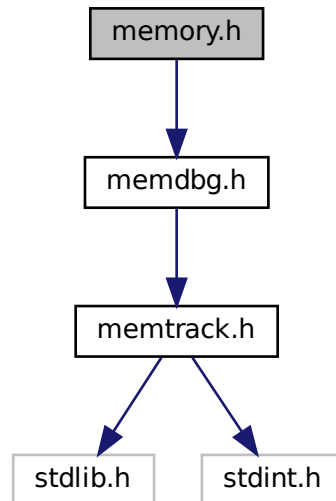
```

5.5 memory.h File Reference

Tracks memory allocation and leaks when compiled without NDEBUG.

```
#include "memdbg.h"
```

Include dependency graph for memory.h:



Macros

- `#define malloc(size) dbg_malloc(size, __FILE__, __LINE__, __DATE__, __TIME__, __func__)`
Same syntax and same behavior than regular malloc function, with memory leaks tracking.
- `#define realloc(ptr, size) dbg_realloc(ptr, size, __FILE__, __LINE__, __DATE__, __TIME__, __func__)`
Same syntax and same behavior than regular realloc function, with memory leaks tracking.
- `#define calloc(nmemb, size) dbg_calloc(nmemb, size, __FILE__, __LINE__, __DATE__, __TIME__, __func__)`
Same syntax and same behavior than regular calloc function, with memory leaks tracking.
- `#define free(ptr) dbg_free(ptr, __FILE__, __LINE__, __DATE__, __TIME__, __func__)`
Same syntax and same behavior than regular free function, with memory leaks tracking.
- `#define strdup(chaine) dbg_strdup(chaine, __FILE__, __LINE__, __DATE__, __TIME__, __func__)`
Same syntax and same behavior than regular strdup function, with memory leaks tracking.
- `#define asprintf(out, format, ...) dbg_asprintf(out, format, __FILE__, __LINE__, __DATE__, __TIME__, __func__, __VA_ARGS__)`
Same syntax and same behavior than regular asprintf function, with memory leaks tracking.
- `#define memreport() memtrack_dumpblocks()`
Prints a list of currently allocated blocks on stderr.

5.5.1 Detailed Description

Tracks memory allocation and leaks when compiled without NDEBUEG.

Date

25/09/2006

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 2017-2024, François Cerbelle

Originally inspired by "L'art du code", Steve Maguire, Microsoft Press This header needs to be included after most of the standard headers, ideally the last one.

Definition in file [memory.h](#).

5.5.2 Macro Definition Documentation

5.5.2.1 asprintf

```
#define asprintf(  
    out,  
    format,  
    ... ) dbg\_asprintf(out, format, __FILE__, __LINE__, __DATE__, __TIME__, __func__, __↵  
VA_ARGS__)
```

Same syntax and same behavior than regular asprintf function, with memory leaks tracking.

Definition at line 47 of file [memory.h](#).

5.5.2.2 calloc

```
#define calloc(  
    nmemb,  
    size ) dbg\_calloc(nmemb, size, __FILE__, __LINE__, __DATE__, __TIME__, __func__)
```

Same syntax and same behavior than regular calloc function, with memory leaks tracking.

Definition at line 38 of file [memory.h](#).

5.5.2.3 free

```
#define free(  
    ptr ) dbg\_free(ptr, __FILE__, __LINE__, __DATE__, __TIME__, __func__)
```

Same syntax and same behavior than regular free function, with memory leaks tracking.

Definition at line 41 of file [memory.h](#).

5.5.2.4 malloc

```
#define malloc(  
    size ) dbg_malloc(size, __FILE__, __LINE__, __DATE__, __TIME__, __func__)
```

Same syntax and same behavior than regular malloc function, with memory leaks tracking.

Definition at line 32 of file [memory.h](#).

5.5.2.5 memreport

```
#define memreport( ) memtrack_dumpblocks()
```

Prints a list of currently allocated blocks on stderr.

Definition at line 50 of file [memory.h](#).

5.5.2.6 realloc

```
#define realloc(  
    ptr,  
    size ) dbg_realloc(ptr, size, __FILE__, __LINE__, __DATE__, __TIME__, __func__)
```

Same syntax and same behavior than regular realloc function, with memory leaks tracking.

Definition at line 35 of file [memory.h](#).

5.5.2.7 strdup

```
#define strdup(  
    chaine ) dbg_strdup(chaine, __FILE__, __LINE__, __DATE__, __TIME__, __func__)
```

Same syntax and same behavior than regular strdup function, with memory leaks tracking.

Definition at line 44 of file [memory.h](#).

5.6 memory.h

[Go to the documentation of this file.](#)

```

00001
00020 #ifndef __MEMORY_H__
00021 #define __MEMORY_H__
00022
00023 #ifdef HAVE_CONFIG_H
00024 #   include "config.h"
00025 #endif
00026
00027 #ifndef NDEBUG
00028
00029 #include "memdbg.h"
00030
00032 #define malloc(size) dbg_malloc(size, __FILE__, __LINE__, __DATE__, __TIME__, __func__)
00033
00035 #define realloc(ptr, size) dbg_realloc(ptr, size, __FILE__, __LINE__, __DATE__, __TIME__, __func__)
00036
00038 #define calloc(nmemb, size) dbg_calloc(nmemb, size, __FILE__, __LINE__, __DATE__, __TIME__, __func__)
00039
00041 #define free(ptr) dbg_free(ptr, __FILE__, __LINE__, __DATE__, __TIME__, __func__)
00042
00044 #define strdup(chaine) dbg_strdup(chaine, __FILE__, __LINE__, __DATE__, __TIME__, __func__)
00045
00047 #define asprintf(out, format, ...)
    dbg_asprintf(out, format, __FILE__, __LINE__, __DATE__, __TIME__, __func__, __VA_ARGS__)
00048
00050 #define memreport() memtrack_dumpblocks()
00051
00052 #else
00053
00055 #define memreport()
00056
00057 #endif
00058 #endif
00059
00060 /* vim: set tw=80: */

```

5.7 memtrack.h File Reference

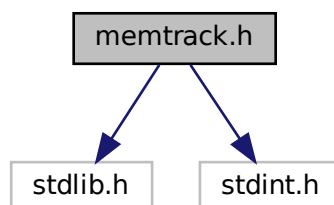
Memory block metadata tracking headers.

```

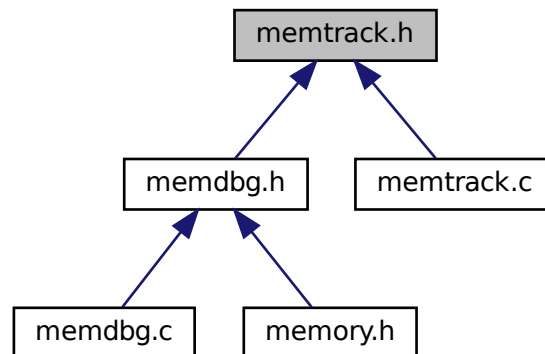
#include <stdlib.h>
#include <stdint.h>

```

Include dependency graph for memtrack.h:



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



Classes

- struct [MemBlock](#)
Memory block metadata list item.

Typedefs

- typedef struct [MemBlock](#) [TMemBlock](#)
Memory block metadata list item.

Variables

- unsigned int(* [memtrack_addblock](#))(const void *p_Ptr, const size_t p_Size, const char *p_File, const int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function)
Functor to register an allocated memory block metadata.
- unsigned int(* [memtrack_delblock](#))(const void *p_Ptr, const char *p_File, const int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function)
Functor to unregister an allocated memory block metadata.
- uint64_t(* [memtrack_dumpblocks](#))()
Functor to list allocated memory blocks metadata.
- uint64_t(* [memtrack_getallocatedblocks](#))()
Functor to get the number of allocated blocks.
- uint64_t(* [memtrack_getallocatedRAM](#))()
Functor to get the total RAM size allocated.
- size_t(* [memtrack_getblocksize](#))(const void *p_Ptr)
Functor to get size of a specific memory block.

5.7.1 Detailed Description

Memory block metadata tracking headers.

Date

25/09/2006

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 1997-2024, François Cerbelle

Originally inspired by "L'art du code", Steve Maguire, Microsoft Press

Definition in file [memtrack.h](#).

5.7.2 Typedef Documentation

5.7.2.1 TMemBlock

```
typedef struct MemBlock TMemBlock
```

Memory block metadata list item.

Double linked list item to store memory block metadata

5.7.3 Variable Documentation

5.7.3.1 memtrack_addblock

```
unsigned int(* memtrack_addblock) (const void *p_Ptr, const size_t p_Size, const char *p_File,  
const int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function) (  
    const void * p_Ptr,  
    const size_t p_Size,  
    const char * p_File,  
    const int p_Line,  
    const char * p_CompilDate,  
    const char * p_CompilTime,  
    const char * p_Function ) [extern]
```

Functor to register an allocated memory block metadata.

Create and adds a memory block metadata record in the tracking system to detect memory leaks. It performs some basic sanity checks. The filename, compilation date and compilation time can not be null or empty, the line number can not be 0, the memory pointer to store can not be NULL or already registered and its size needs to be greater than 0.

The function is called from [memdbg.c](#) functions used in the application code thru [memory.h](#) macros. The macro automatically fills the filename, line number, compilation date and time, and function name (using GCC's non-ansi `__func__` extension).

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Parameters

in	<i>p_Ptr</i>	Allocated memory block pointer
in	<i>p_Size</i>	Allocated memory block size
in	<i>p_File</i>	: Source file
in	<i>p_Line</i>	: Source line number
in	<i>p_CompilDate</i>	: File compilation date
in	<i>p_CompilTime</i>	: File compilation time
in	<i>p_Function</i>	: Source function name

Returns

Registration status

Return values

0	if succeeded,
1	if not possible.

Definition at line 584 of file [memtrack.c](#).

5.7.3.2 memtrack_delblock

```
unsigned int(* memtrack_delblock) (const void *p_Ptr, const char *p_File, const int p_Line,
const char *p_CompilDate, const char *p_CompilTime, const char *p_Function) (
    const void * p_Ptr,
    const char * p_File,
    const int p_Line,
    const char * p_CompilDate,
    const char * p_CompilTime,
    const char * p_Function ) [extern]
```

Functor to unregister an allocated memory block metadata.

Find and delete a memory block metadata record in the tracking system. It performs some basic sanity checks. The filename, compilation date and compilation time can not be null or empty, the line number can not be 0, the memory pointer to remove can not be NULL, it needs to already be registered.

The function is called from [memdbg.c](#) functions used in the application code thru [memory.h](#) macros. The macro automatically fills the filename, line number, compilation date and time, and function name (using GCC's non-ansi `__func__` extension).

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Parameters

in	<i>p_Ptr</i>	Allocated memory block pointer
in	<i>p_File</i>	: Source file
in	<i>p_Line</i>	: Source line number
in	<i>p_CompilDate</i>	: File compilation date
in	<i>p_CompilTime</i>	: File compilation time
in	<i>p_Function</i>	: Source function name

Returns

Registration status

Return values

0	if succeeded,
!0	if not possible.

Definition at line 621 of file [memtrack.c](#).

5.7.3.3 memtrack_dumpblocks

```
uint64_t(* memtrack_dumpblocks) () ( ) [extern]
```

Functor to list allocated memory blocks metadata.

Dumps all the metadata of the registered memory blocks to stderr.

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Returns

Number of registered blocks (counted)

Return values

0	if succeeded,
!0	if not possible.

Definition at line 642 of file [memtrack.c](#).

5.7.3.4 memtrack_getallocatedblocks

```
uint64_t(* memtrack_getallocatedblocks) () ( ) [extern]
```

Functor to get the number of allocated blocks.

This function returns the value of the internal counter of allocated memory blocks metadata.

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Returns

Number of registered blocks

Definition at line 656 of file [memtrack.c](#).

5.7.3.5 memtrack_getallocatedRAM

```
uint64_t(* memtrack_getallocatedRAM) () ( ) [extern]
```

Functor to get the total RAM size allocated.

This function returns the internal summ of all the allocated memory blocks which are registered.

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Returns

Total RAM size in bytes

Definition at line 670 of file [memtrack.c](#).

5.7.3.6 memtrack_getblocksize

```
size_t(* memtrack_getblocksize) (const void *p_Ptr) (
    const void * p_Ptr ) [extern]
```

Functor to get size of a specific memory block.

The function will search in the list for the specified pointer. If the pointer is not found, it will return 0, which is discriminant as the memtracker does not allow to track a zero sized block. The memtracker does not allow neither to track a NULL pointer, thus NULL will return 0. Otherwise, the function will return the memory block size.

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Parameters

in	<i>p_Ptr</i>	Allocated and tracked memory block pointer
----	--------------	--

Returns

Memory block size in bytes

Definition at line 689 of file [memtrack.c](#).

5.8 memtrack.h

[Go to the documentation of this file.](#)

```
00001
00019 #ifndef __MEMTRACK_H__
00020 #define __MEMTRACK_H__
```



```

00021
00022 #ifdef HAVE_CONFIG_H
00023 # include "config.h"
00024 #endif
00025
00026 #include <stdlib.h>          /* size_t */
00027 #include <stdint.h>         /* uint64_t */
00028
00033 typedef struct MemBlock {
00034     struct MemBlock *Prev;
00035     struct MemBlock *Next;
00036     void *Ptr;
00037     size_t Size;
00038     char *File;
00039     int Line;
00040     char *CompileDate;
00041     char *CompileTime;
00042     char *Function;
00043 } TMemBlock;
00044
00045 extern unsigned int (*memtrack_addblock) (const void *p_Ptr,
00046     const size_t p_Size,
00047     const char *p_File,
00048     const int p_Line,
00049     const char *p_CompileDate,
00050     const char *p_CompileTime,
00051     const char *p_Function);
00052 extern unsigned int (*memtrack_delblock) (const void *p_Ptr,
00053     const char *p_File,
00054     const int p_Line,
00055     const char *p_CompileDate,
00056     const char *p_CompileTime,
00057     const char *p_Function);
00058 extern uint64_t (*memtrack_dumpblocks) ();
00059 extern uint64_t (*memtrack_getallocatedblocks) ();
00060 extern uint64_t (*memtrack_getallocatedRAM) ();
00061 extern size_t (*memtrack_getblocksize) (const void *p_Ptr);
00062
00063 #endif          /* __MEMTRACK_H__ */
00064
00065 /* vim: set tw=80: */

```

5.9 assert.c File Reference

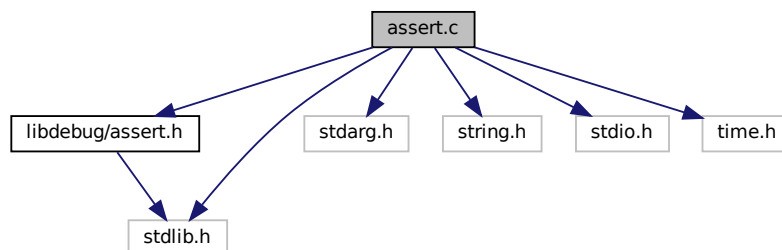
Compiled functions used by debugging macros to write on stderr.

```

#include "libdebug/assert.h"
#include <stdarg.h>
#include <string.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

```

Include dependency graph for assert.c:



Macros

- `#define _XOPEN_SOURCE 500 /* vasprintf */`

Functions

- void `_trace` (const char *p_File, const unsigned int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function)
Print a debug trace (checkpoint)
- void `_trace_msg` (const char *p_File, const unsigned int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function, const char *p_Message)
Print a debug trace (checkpoint) with a static message.
- void `_trace_dynmsg` (const char *p_File, const unsigned int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function, const char *p_Format,...)
Print a debug trace (checkpoint) with a formatted message.

5.9.1 Detailed Description

Compiled functions used by debugging macros to write on stderr.

Date

11/05/1997

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 1997-2024, François Cerbelle

Originally inspired by "L'art du code", Steve Maguire, Microsoft Press

Definition in file [assert.c](#).

5.9.2 Macro Definition Documentation

5.9.2.1 _XOPEN_SOURCE

```
#define _XOPEN_SOURCE 500 /* vasprintf */
```

Definition at line 20 of file [assert.c](#).

5.9.3 Function Documentation

5.9.3.1 `_trace()`

```
void _trace (
    const char * p_File,
    const unsigned int p_Line,
    const char * p_CompilDate,
    const char * p_CompilTime,
    const char * p_Function )
```

Print a debug trace (checkpoint)

Parameters

in	<i>p_File</i>	Source file
in	<i>p_Line</i>	Source line in the source file
in	<i>p_CompilDate</i>	Compilation date
in	<i>p_CompilTime</i>	Compilation time
in	<i>p_Function</i>	Function name in the source file

Outputs on stderr a timestamp, with the filename, the sourceline, the compilation date and time, the function name.

Definition at line 77 of file [assert.c](#).

5.9.3.2 `_trace_dynmsg()`

```
void _trace_dynmsg (
    const char * p_File,
    const unsigned int p_Line,
    const char * p_CompilDate,
    const char * p_CompilTime,
    const char * p_Function,
    const char * p_Format,
    ... )
```

Print a debug trace (checkpoint) with a formatted message.

Parameters

in	<i>p_File</i>	Source file
in	<i>p_Line</i>	Source line in the source file
in	<i>p_CompilDate</i>	Compilation date
in	<i>p_CompilTime</i>	Compilation time
in	<i>p_Function</i>	Function name in the source file
in	<i>p_Format</i>	format string
in	...	Formatted string parameters

Outputs on stderr a timestamp, with the filename, the sourceline, the compilation date and time, the function name and a formatted message.

Todo Replace with a portable snprintf function

Definition at line 101 of file [assert.c](#).

5.9.3.3 _trace_msg()

```
void _trace_msg (
    const char * p_File,
    const unsigned int p_Line,
    const char * p_CompilDate,
    const char * p_CompilTime,
    const char * p_Function,
    const char * p_Message )
```

Print a debug trace (checkpoint) with a static message.

Parameters

in	<i>p_File</i>	Source file
in	<i>p_Line</i>	Source line in the source file
in	<i>p_CompilDate</i>	Compilation date
in	<i>p_CompilTime</i>	Compilation time
in	<i>p_Function</i>	Function name in the source file
in	<i>p_Message</i>	Static message

Outputs on stderr a timestamp, with the filename, the sourceline, the compilation date and time, the function name and a static message.

Definition at line 88 of file [assert.c](#).

5.10 assert.c

[Go to the documentation of this file.](#)

```
00001
00020 #define _XOPEN_SOURCE 500                                /* vasprintf */
00021 #include "libdebug/assert.h"
00022 #include <stdarg.h>                                        /* va_list, va_start, va_arg, va_end */
00023 #include <string.h>                                       /* strcpy */
00024 #include <stdio.h>                                        /* fprintf */
00025 #include <time.h>                                         /* time, localtime */
00026 #include <stdlib.h>                                       /* abort */
00027 #include <stdio.h>                                       /* snprintf */
00028
00044 static const char*_timestamp(const char* p_File,
00045                               const unsigned int p_Line,
00046                               const char* p_CompilDate,
00047                               const char* p_CompilTime,
00048                               const char* p_Function) {
00049     /* Get local time and format it */
00050     char l_Time[24];
00051     static char l_tmp[120];
```

```

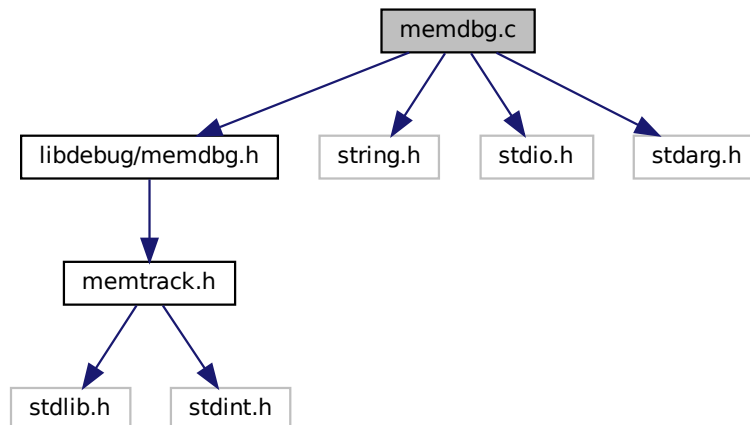
00052     time_t l_CurrentTime = time(NULL);
00053
00054     /* Parameter validity check against invalid parameters such as NULL or
00055     * empty string values. */
00056     if ((NULL==p_File)|| (0==p_File[0])||
00057         (0==p_Line)||
00058         (NULL==p_CompilDate)|| (0==p_CompilDate[0])||
00059         (NULL==p_CompilTime)|| (0==p_CompilTime[0])||
00060         (NULL==p_Function)|| (0==p_Function[0])) {
00061         fprintf(stderr,"%s:%d Unexpected and invalid parameters\n",__FILE__, __LINE__);
00062         abort();
00063     }
00064
00065     strftime(l_Time, sizeof(l_Time), "%Y-%m-%d %H:%M:%S", localtime(&l_CurrentTime));
00066
00067     /* Timestamp build with the filename, file line, function name */
00068     snprintf(l_tmp,sizeof(l_tmp),
00069              "%19s [%20s:%-4ud] (%11s @ %8s) %30s()",
00070              l_Time,p_File,p_Line,p_CompilDate,p_CompilTime,p_Function /* cppcheck-suppress
00071              ctunullpointer */
00072              );
00073     return &l_tmp[0];
00074 }
00075
00076 /* Documentation in header file */
00077 void _trace(const char* p_File,
00078            const unsigned int p_Line,
00079            const char* p_CompilDate,
00080            const char* p_CompilTime, const char* p_Function) {
00081     /* Parameter validity enforced by _timestamp */
00082     fprintf (stderr,"%s\n",
00083             _timestamp( p_File, p_Line, p_CompilDate, p_CompilTime, p_Function)
00084             );
00085 }
00086
00087 /* Documentation in header file */
00088 void _trace_msg(const char* p_File,
00089                const unsigned int p_Line,
00090                const char* p_CompilDate,
00091                const char* p_CompilTime,
00092                const char* p_Function, const char* p_Message) {
00093     /* Parameter validity enforced by _timestamp */
00094     fprintf (stderr,"%s : %s\n",
00095             _timestamp( p_File, p_Line, p_CompilDate, p_CompilTime, p_Function),
00096             p_Message /* cppcheck-suppress ctunullpointer */
00097             );
00098 }
00099
00100 /* Documentation in header file */
00101 void _trace_dynmsg(const char* p_File,
00102                   const unsigned int p_Line,
00103                   const char* p_CompilDate,
00104                   const char* p_CompilTime,
00105                   const char* p_Function, const char* p_Format, ...
00106                   ) {
00107     /* Formatted message string */
00108     char l_tmp[100];
00109     /* Formatted message length */
00110     unsigned int l_length;
00111
00112     /* Limit variadic processing scope in a block */
00113     {
00114         /* Build the formatted message */
00115         va_list l_ap;
00116         va_start(l_ap, p_Format);
00117         l_length = vsnprintf(l_tmp,sizeof(l_tmp),p_Format,l_ap);
00118         va_end(l_ap);
00119     }
00120
00121     if (l_length >= sizeof(l_tmp)-1) {
00122         /* Indicate that message was truncated */
00123         strcpy(&l_tmp[sizeof(l_tmp)-6]), "...");
00124     }
00125
00126     /* Parameter validity enforced by _timestamp */
00127     fprintf (stderr,"%s : %s\n",
00128             _timestamp( p_File, p_Line, p_CompilDate, p_CompilTime, p_Function),
00129             l_tmp
00130             );
00131 }
00132
00133 /* vim: set tw=80: */

```

5.11 memdbg.c File Reference

Memory leak tracker implementation.

```
#include "libdebug/memdbg.h"
#include <string.h>
#include <stdio.h>
#include <stdarg.h>
Include dependency graph for memdbg.c:
```



Macros

- `#define _XOPEN_SOURCE 500 /* strdup */`
- `#define _GNU_SOURCE /* vasprintf */`

Functions

- void * [dbg_malloc](#) (const size_t Size, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function)
Malloc compatible standard allocation.
- void [dbg_free](#) (void *Ptr, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function)
Free compatible standard memory release.
- void * [dbg_calloc](#) (const size_t NMem, const size_t Size, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function)
Allocate a table of item from the size of each and number.
- void * [dbg_realloc](#) (void *Ptr, const size_t Size, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function)
Resize an already allocated and tracked block.
- char * [dbg_strdup](#) (const char *Ptr, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function)
String duplication with allocation.
- int [dbg_asprintf](#) (char **p_Ptr, const char *p_Format, const char *File, const int Line, const char *CompilDate, const char *CompilTime, const char *Function,...)
Build a formatted string with allocation.

5.11.1 Detailed Description

Memory leak tracker implementation.

Date

25/09/2006

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 1997-2024, François Cerbelle

Originally inspired by "L'art du code", Steve Maguire, Microsoft Press

Definition in file [memdbg.c](#).

5.11.2 Macro Definition Documentation

5.11.2.1 _GNU_SOURCE

```
#define _GNU_SOURCE /* vasprintf */
```

Definition at line 20 of file [memdbg.c](#).

5.11.2.2 _XOPEN_SOURCE

```
#define _XOPEN_SOURCE 500 /* strdup */
```

Definition at line 19 of file [memdbg.c](#).

5.11.3 Function Documentation

5.11.3.1 dbg_asprintf()

```
int dbg_asprintf (  
    char ** p_Ptr,  
    const char * p_Format,  
    const char * File,  
    const int Line,  
    const char * CompilDate,  
    const char * CompilTime,  
    const char * Function,  
    ... )
```

Build a formatted string with allocation.

Parameters

in, out	<i>p_Ptr</i>	: Pointer on the to be built string
in	<i>p_Format</i>	: Pointer on the format string
in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompilDate</i>	: File compilation date
in	<i>CompilTime</i>	: File compilation time
in	<i>Function</i>	: Source function name
in	...	: Values referenced by the format string

Returns

Status of the formatting

Return values

≥ 0	number of chars in the output string (as strdup)
-1	in case of error (*p_Ptr contents is undefined)

Todo Implement a vasprintf wrapping function to catch allocation and use it here

Definition at line 200 of file [memdbg.c](#).

5.11.3.2 dbg_calloc()

```
void * dbg_calloc (
    const size_t NMemb,
    const size_t Size,
    const char * File,
    const int Line,
    const char * CompilDate,
    const char * CompilTime,
    const char * Function )
```

Allocate a table of item from the size of each and number.

Uses malloc to allocate and track the memory bloc. If allocation and tracking succeed, fill the memory block with zeros.

Parameters

in	<i>NMemb</i>	: Item number in the table
in	<i>Size</i>	: Item size in bytes
in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompilDate</i>	: File compilation date
in	<i>CompilTime</i>	: File compilation time
in	<i>Function</i>	: Source function name

Returns

Allocated block address

Return values

<i>Address</i>	if succeeded,
<i>NULL</i>	if not possible.

Definition at line 83 of file [memdbg.c](#).

Here is the call graph for this function:

**5.11.3.3 dbg_free()**

```

void dbg_free (
    void * Ptr,
    const char * File,
    const int Line,
    const char * CompilDate,
    const char * CompilTime,
    const char * Function )
  
```

Free compatible standard memory release.

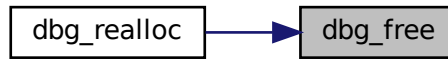
If the *Ptr* value is not NULL, try to untrack it. If it can not be found in the tracked list, because it was not tracked, it was allocated by a non monitored function, the function abort the process for investigation (missing a monitoring macro/function in the tracker, bug in the tracker, allocation from an external non-instrumented library). If the *Ptr* value was tracked, found and removed from the list successfully, forward it to free for actual free.

Parameters

in	<i>Ptr</i>	: Pointer on the memory to free
in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompilDate</i>	: File compilation date
in	<i>CompilTime</i>	: File compilation time
in	<i>Function</i>	: Source function name

Definition at line 60 of file [memdbg.c](#).

Here is the caller graph for this function:



5.11.3.4 dbg_malloc()

```

void * dbg_malloc (
    const size_t Size,
    const char * File,
    const int Line,
    const char * CompilDate,
    const char * CompilTime,
    const char * Function )
  
```

Malloc compatible standard allocation.

Parameters

in	<i>Size</i>	: Requested size in bytes
in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompilDate</i>	: File compilation date
in	<i>CompilTime</i>	: File compilation time
in	<i>Function</i>	: Source function name

Returns

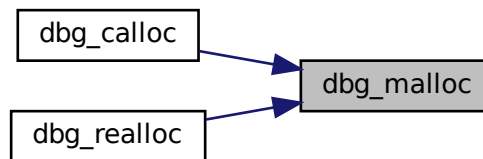
Allocated block address

Return values

<i>Address</i>	if succeeded,
<i>NULL</i>	if not possible.

Definition at line 27 of file [memdbg.c](#).

Here is the caller graph for this function:



5.11.3.5 dbg_realloc()

```

void * dbg_realloc (
    void * Ptr,
    const size_t Size,
    const char * File,
    const int Line,
    const char * CompileDate,
    const char * CompileTime,
    const char * Function )
  
```

Resize an already allocated and tracked block.

This function always uses malloc to allocate a new block and force an address change, and a data loss in case of shrink, which is the worst case scenario for realloc.

As for realloc, a NULL source pointer makes realloc act as malloc, and a new size set to 0 acts as free. The input pointer is left untouched but should not be used or freed anymore. It is always different than the return value. The input values referenced by the input pointer are not wiped.

This implementation can not be used if realloc is used to reduce a huge bloc in order to manage an OOM situation. Real realloc can succeed by actually downsizing the same memory block, inplace, but this implementation will fail because it first allocate a new block.

Parameters

in	<i>Ptr</i>	: Pointer on the memory to resize
in	<i>Size</i>	: New size in bytes
in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompileDate</i>	: File compilation date
in	<i>CompileTime</i>	: File compilation time
in	<i>Function</i>	: Source function name

Returns

Resized block address

Return values

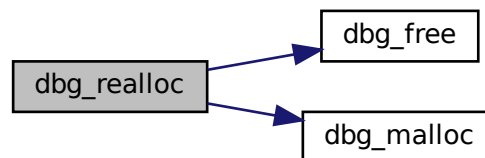
<i>Address</i>	if succeeded,
<i>NULL</i>	if not possible.

< Existing bloc size

< New block

Definition at line 108 of file [memdbg.c](#).

Here is the call graph for this function:

**5.11.3.6 dbg_strdup()**

```

char * dbg_strdup (
    const char * Ptr,
    const char * File,
    const int Line,
    const char * CompilDate,
    const char * CompilTime,
    const char * Function )
  
```

String duplication with allocation.

Parameters

in	<i>Ptr</i>	: Pointer on the string to copy
in	<i>File</i>	: Source file
in	<i>Line</i>	: Source line number
in	<i>CompilDate</i>	: File compilation date
in	<i>CompilTime</i>	: File compilation time
in	<i>Function</i>	: Source function name

Returns

Copied string address

Return values

Address	if succeeded,
NULL	if not possible.

Copy address

Definition at line 168 of file [memdbg.c](#).

5.12 memdbg.c

[Go to the documentation of this file.](#)

```

00001
00019 #define _XOPEN_SOURCE 500                                /* strdup */
00020 #define _GNU_SOURCE                                       /* vasprintf */
00021 #include "libdebug/memdbg.h"
00022 #include <string.h>                                       /* memset, memcpy, memmove */
00023 #include <stdio.h>                                        /* asprintf */
00024 #include <stdarg.h>                                       /* va_list, va_start, va_arg, va_end */
00025
00026 /* Documented in header file */
00027 void* dbg_malloc(
00028     const size_t Size,
00029     const char* File,
00030     const int Line,
00031     const char* CompileDate,
00032     const char* CompileTime,
00033     const char* Function
00034 ) {
00035     /* Memory allocation */
00036     void* l_tmp = malloc(Size);
00037     if (NULL == l_tmp) return (l_tmp);
00038
00039     /* If successful, track the memory block */
00040 #pragma GCC diagnostic push                                /* save the actual diag context */
00041 #ifdef __clang__
00042 #elif __GNUC__
00043 #pragma GCC diagnostic ignored "-Wmaybe-uninitialized" /* locally disable maybe warnings */
00044 #elif _MSC_VER
00045     /*usually has the version number in _MSC_VER*/
00046 #elif _BORLANDC__
00047 #elif _MINGW32__
00048 #endif
00049     if (0 != memtrack_addblock( l_tmp, Size, File, Line, CompileDate, CompileTime, Function)) {
00050 #pragma GCC diagnostic pop                                /* restore previous diag context */
00051     /* If tracking fails, the whole allocation fails */
00052     free(l_tmp);
00053     l_tmp = NULL;
00054     };
00055     return l_tmp;
00056 }
00057
00058
00059 /* Documented in header file */
00060 void dbg_free(
00061     void* Ptr,
00062     const char* File,
00063     const int Line,
00064     const char* CompileDate,
00065     const char* CompileTime,
00066     const char* Function
00067 ) {
00068     /* If the pointer was not NULL, it was tracked, remove it from the tracked
00069     * list. If it was not tracked, removing returns an error. Abort the
00070     * process as it should never have an untracked pointer. Either it was
00071     * allocated from a non instrumented binary, or it was allocated from
00072     * a non monitored function (see memory.h) or there is a bug in the
00073     * memory leak tracker. */
00074     if (NULL != Ptr)

```

```

00075         if (0!=memtrack_delblock(Ptr,File,Line,CompilDate,CompilTime,Function))
00076             abort();
00077
00078     /* If the pointer was NULL or tracked, forward it to the real free */
00079     free(Ptr);
00080 }
00081
00082 /* Documented in header file */
00083 void* dbg_calloc(
00084     const size_t NMem,
00085     const size_t Size,
00086     const char* File,
00087     const int Line,
00088     const char* CompilDate,
00089     const char* CompilTime,
00090     const char* Function
00091 ) {
00092     void* l_tmp;
00093
00094     /* Use the dbg_malloc function to allocate the memory */
00095     l_tmp = dbg_malloc(
00096         NMem*Size,
00097         File,Line,CompilDate,CompilTime,Function);
00098
00099     /* Implement the calloc specific behavior compared to simple malloc:
00100     * it fills the allocated memory block with 0 */
00101     if (NULL != l_tmp)
00102         memset((char*)l_tmp, 0, NMem*Size);
00103
00104     return l_tmp;
00105 }
00106
00107 /* Documented in header file */
00108 void* dbg_realloc(
00109     void* Ptr,
00110     const size_t Size,
00111     const char* File,
00112     const int Line,
00113     const char* CompilDate,
00114     const char* CompilTime,
00115     const char* Function
00116 ) {
00117     size_t l_oldsize;
00118     char *newblk;
00119     /* NULL is not tracked but valid */
00120     if (NULL==Ptr) {
00121         l_oldsize=0;
00122     } else {
00123         /* Fetch existing block size */
00124         l_oldsize = memtrack_getblocksize(Ptr);
00125
00126         /* This is probably a bug in the memory tracker.
00127         * It should not track zero sized blocks */
00128         if (0==l_oldsize)
00129             abort();
00130     }
00131
00132     /* If new size is 0, then act as free, like realloc */
00133     if (0==Size) {
00134         dbg_free(Ptr,
00135             File,Line,CompilDate,CompilTime,Function);
00136         return Ptr;
00137     }
00138
00139     /* New sized block allocation to simulate the worst case scenario and
00140     * test a pointer change, a data loss (in case of shrink) */
00141     newblk=(char*)dbg_malloc(
00142         Size,
00143         File,Line,CompilDate,CompilTime,Function);
00144
00145     /* The new block can fail */
00146     /* The real realloc function could succeed here, in case of inplace
00147     * shrink in an OOM situation. */
00148     if (NULL == newblk) return (newblk);
00149
00150     /* Copy only the relevant data from old block to new block, losing extra
00151     * data in case of shrink, and not initializing new data in case of
00152     * increase */
00153     memcpy(newblk, (char*)Ptr, (l_oldsize<Size?l_oldsize:Size));
00154
00155     /* Free old block */
00156     dbg_free(
00157         Ptr,
00158         File,Line,CompilDate,CompilTime,Function);
00159
00160     return newblk;
00161 }
00162

```

```

00163     return (void*) newblk;
00164 }
00165
00166
00167 /* Documented in header file */
00168 char* dbg_strdup(
00169     const char* Ptr,
00170     const char* File,
00171     const int Line,
00172     const char* CompilDate,
00173     const char* CompilTime,
00174     const char* Function
00175 ) {
00176     char* l_newblk;
00177     /* Use strdup to actually copy the string with its own return values and
00178  * abort (SIGSEGV in case of NULL) */
00179     l_newblk=NULL;
00180     l_newblk=strdup(Ptr); /* cppcheck-suppress ctunullpointer */
00181
00182     /* If the copy succeeded, try to track the memory allocation */
00183     if (NULL != l_newblk)
00184         if (0!=memtrack_addblock(
00185             l_newblk,
00186             strlen(l_newblk)+1,
00187             File,Line,CompilDate,CompilTime,Function
00188         )) {
00189             /* If tracking fails, the whole operation is reverted and fails */
00190             free(l_newblk);
00191             l_newblk=NULL;
00192         }
00193     };
00194
00195     return l_newblk;
00196 }
00197
00198
00199 /* Documented in header file */
00200 int dbg_asprintf(char **p_Ptr,
00201     const char* p_Format,
00202     const char *File,
00203     const int Line,
00204     const char *CompilDate,
00205     const char *CompilTime, const char *Function,
00206     ...) {
00207     int l_returncode;
00208
00209     /* NULL is not allowed, where would we store the result, then ? */
00210     if (NULL==p_Ptr)
00211         abort();
00212
00213     /* Limit the scope of the variadic manipulation variables */
00214     {
00215         va_list l_ap;
00216         va_start (l_ap, Function);
00217         /* Use the original vasprintf to build the formatted string */
00220         l_returncode = vasprintf(p_Ptr, p_Format, l_ap);
00221         va_end(l_ap);
00222     }
00223
00224     /* If formatting succeeded, try to track the memory allocation */
00225     if (-1 != l_returncode)
00226         if (1==memtrack_addblock(
00227             *p_Ptr,
00228             strlen (*p_Ptr)+1,
00229             File,Line,CompilDate,CompilTime,Function
00230         )) {
00231             /* If tracking fails, the whole operation is reverted and fails */
00232             free(*p_Ptr);
00233             l_returncode=-2;
00234         }
00235     };
00236
00237     return l_returncode;
00238 }
00239
00240 /* vim: set tw=80: */

```

5.13 memtrack.c File Reference

Memory block metadata tracking implementation.

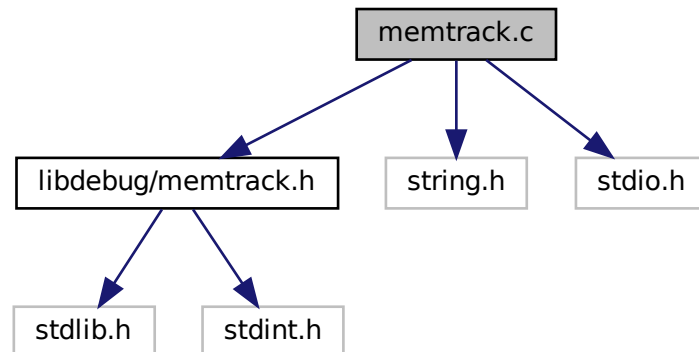
```

#include "libdebug/memtrack.h"
#include <string.h>

```

```
#include <stdio.h>
```

Include dependency graph for memtrack.c:



Macros

- `#define _XOPEN_SOURCE 500 /* strdup */`

Functions

- void `memtrack_reset()`
Memory block metadata list reset.

Variables

- unsigned int(* `memtrack_addblock`)(const void *p_Ptr, const size_t p_Size, const char *p_File, const int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function)
Functor to register an allocated memory block metadata.
- unsigned int(* `memtrack_delblock`)(const void *p_Ptr, const char *p_File, const int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function)
Functor to unregister an allocated memory block metadata.
- uint64_t(* `memtrack_dumpblocks`)() = `memtrack_dumpblocks_preinit`
Functor to list allocated memory blocks metadata.
- uint64_t(* `memtrack_getallocatedblocks`)() = `memtrack_getallocatedblocks_preinit`
Functor to get the number of allocated blocks.
- uint64_t(* `memtrack_getallocatedRAM`)() = `memtrack_getallocatedRAM_preinit`
Functor to get the total RAM size allocated.
- size_t(* `memtrack_getblocksize`)(const void *p_Ptr) = `memtrack_getblocksize_preinit`
Functor to get size of a specific memory block.

5.13.1 Detailed Description

Memory block metadata tracking implementation.

Date

25/09/2006

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 1997-2024, François Cerbelle

Originally inspired by "L'art du code", Steve Maguire, Microsoft Press

Definition in file [memtrack.c](#).

5.13.2 Macro Definition Documentation

5.13.2.1 _XOPEN_SOURCE

```
#define _XOPEN_SOURCE 500 /* strdup */
```

Definition at line 19 of file [memtrack.c](#).

5.13.3 Function Documentation

5.13.3.1 memtrack_reset()

```
void memtrack_reset ( )
```

Memory block metadata list reset.

This function is not defined in the header file because it should never be used. It is exported because it is used by the unit tests. NEVER USE IT, it will crash your code.

Definition at line 75 of file [memtrack.c](#).

5.13.4 Variable Documentation

5.13.4.1 memtrack_addblock

```
unsigned int(* memtrack_addblock) (const void *p_Ptr, const size_t p_Size, const char *p_File,
const int p_Line, const char *p_CompilDate, const char *p_CompilTime, const char *p_Function) (
    const void * p_Ptr,
    const size_t p_Size,
    const char * p_File,
    const int p_Line,
    const char * p_CompilDate,
    const char * p_CompilTime,
    const char * p_Function )
```

Initial value:

```
=
                                memtrack_addblock_preinit
```

Functor to register an allocated memory block metadata.

Create and adds a memory block metadata record in the tracking system to detect memory leaks. It performs some basic sanity checks. The filename, compilation date and compilation time can not be null or empty, the line number can not be 0, the memory pointer to store can not be NULL or already registered and its size needs to be greater than 0.

The function is called from [memdbg.c](#) functions used in the application code thru [memory.h](#) macros. The macro automatically fills the filename, line number, compilation date and time, and function name (using GCC's non-ansi `__func__` extension).

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Parameters

in	<i>p_Ptr</i>	Allocated memory block pointer
in	<i>p_Size</i>	Allocated memory block size
in	<i>p_File</i>	: Source file
in	<i>p_Line</i>	: Source line number
in	<i>p_CompilDate</i>	: File compilation date
in	<i>p_CompilTime</i>	: File compilation time
in	<i>p_Function</i>	: Source function name

Returns

Registration status

Return values

0	if succeeded,
1	if not possible.

Definition at line 584 of file [memtrack.c](#).

5.13.4.2 memtrack_delblock

```
unsigned int(* memtrack_delblock) (const void *p_Ptr, const char *p_File, const int p_Line,
const char *p_CompilDate, const char *p_CompilTime, const char *p_Function) (
    const void * p_Ptr,
    const char * p_File,
    const int p_Line,
    const char * p_CompilDate,
    const char * p_CompilTime,
    const char * p_Function )
```

Initial value:

```
=
                                memtrack_delblock_preinit
```

Functor to unregister an allocated memory block metadata.

Find and delete a memory block metadata record in the tracking system. It performs some basic sanity checks. The filename, compilation date and compilation time can not be null or empty, the line number can not be 0, the memory pointer to remove can not be NULL, it needs to already be registered.

The function is called from [memdbg.c](#) functions used in the application code thru [memory.h](#) macros. The macro automatically fills the filename, line number, compilation date and time, and function name (using GCC's non-ansi `__func__` extension).

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Parameters

in	<i>p_Ptr</i>	Allocated memory block pointer
in	<i>p_File</i>	: Source file
in	<i>p_Line</i>	: Source line number
in	<i>p_CompilDate</i>	: File compilation date
in	<i>p_CompilTime</i>	: File compilation time
in	<i>p_Function</i>	: Source function name

Returns

Registration status

Return values

0	if succeeded,
!0	if not possible.

Definition at line 621 of file [memtrack.c](#).

5.13.4.3 memtrack_dumpblocks

```
uint64_t(* memtrack_dumpblocks) () ( ) = memtrack_dumpblocks_preinit
```

Functor to list allocated memory blocks metadata.

Dumps all the metadata of the registered memory blocks to stderr.

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Returns

Number of registered blocks (counted)

Return values

0	if succeeded,
!0	if not possible.

Definition at line 642 of file [memtrack.c](#).

5.13.4.4 memtrack_getallocatedblocks

```
uint64_t(* memtrack_getallocatedblocks) () ( ) = memtrack_getallocatedblocks_preinit
```

Functor to get the number of allocated blocks.

This function returns the value of the internal counter of allocated memory blocks metadata.

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Returns

Number of registered blocks

Definition at line 656 of file [memtrack.c](#).

5.13.4.5 memtrack_getallocatedRAM

```
uint64_t(* memtrack_getallocatedRAM) () ( ) = memtrack_getallocatedRAM_preinit
```

Functor to get the total RAM size allocated.

This function returns the internal summ of all the allocated memory blocks which are registered.

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Returns

Total RAM size in bytes

Definition at line 670 of file [memtrack.c](#).

5.13.4.6 memtrack_getblocksize

```
size_t(* memtrack_getblocksize) (const void *p_Ptr) (
    const void * p_Ptr ) = memtrack_getblocksize_preinit
```

Functor to get size of a specific memory block.

The function will search in the list for the specified pointer. If the pointer is not found, it will return 0, which is discriminant as the memtracker does not allow to track a zero sized block. The memtracker does not allow neither to track a NULL pointer, thus NULL will return 0. Otherwise, the function will return the memory block size.

This functor is used to implement a lazy initialization. It initially reference a temporary function to trigger memtrack initialization before calling the actual function. Then, it references the actual function directly to avoid any useless tests.

Parameters

in	<i>p_Ptr</i>	Allocated and tracked memory block pointer
----	--------------	--

Returns

Memory block size in bytes

Definition at line 689 of file [memtrack.c](#).

5.14 memtrack.c

[Go to the documentation of this file.](#)

```
00001
00019 #define _XOPEN_SOURCE 500                                /* strdup */
00020 #include "libdebug/memtrack.h"
```

```

00021 #include <string.h>                                /* strdup */
00022 #include <stdio.h>                                    /* fprintf */
00023
00024 static TMemBlock *Head;
00025 static TMemBlock *Tail;
00026 static uint64_t NbBlocks;
00027 static uint64_t RAMSize;
00029 static unsigned int memtrack_init();
00030 static unsigned int memtrack_addblock_preinit(const void *p_Ptr,
00031     const size_t p_Size,
00032     const char *p_File,
00033     const int p_Line,
00034     const char *p_CompilDate,
00035     const char *p_CompilTime,
00036     const char *p_Function);
00037 static unsigned int memtrack_delblock_preinit(const void *p_Ptr,
00038     const char *p_File,
00039     const int p_Line,
00040     const char *p_CompilDate,
00041     const char *p_CompilTime,
00042     const char *p_Function);
00043 static uint64_t memtrack_dumpblocks_preinit();
00044 static uint64_t memtrack_getallocatedblocks_preinit();
00045 static uint64_t memtrack_getallocatedRAM_preinit();
00046 static size_t memtrack_getblocksize_preinit(const void *p_Ptr);
00047 static unsigned int memtrack_addblock_postinit(const void *p_Ptr,
00048     const size_t p_Size,
00049     const char *p_File,
00050     const int p_Line,
00051     const char *p_CompilDate,
00052     const char *p_CompilTime,
00053     const char *p_Function);
00054 static unsigned int memtrack_delblock_postinit(const void *p_Ptr,
00055     const char *p_File,
00056     const int p_Line,
00057     const char *p_CompilDate,
00058     const char *p_CompilTime,
00059     const char *p_Function);
00060 static uint64_t memtrack_dumpblocks_postinit();
00061 static uint64_t memtrack_getallocatedblocks_postinit();
00062 static uint64_t memtrack_getallocatedRAM_postinit();
00063 static size_t memtrack_getblocksize_postinit(const void *p_Ptr);
00064
00065 /*****
00066  * Implementations */
00067 *****/
00068
00075 void memtrack_reset() {
00076     /* Ne réinitialise que si nécessaire */
00077     if ( memtrack_addblock == memtrack_addblock_preinit)
00078         return;
00079
00080     /* Déconfiguration des foncteurs */
00081     memtrack_addblock = memtrack_addblock_preinit;
00082     memtrack_delblock = memtrack_delblock_preinit;
00083     memtrack_dumpblocks = memtrack_dumpblocks_preinit;
00084     memtrack_getallocatedblocks = memtrack_getallocatedblocks_preinit;
00085     memtrack_getallocatedRAM = memtrack_getallocatedRAM_preinit;
00086     memtrack_getblocksize = memtrack_getblocksize_preinit;
00087
00088     /* Purge de la liste */
00089     while (Tail!=Head->Next) {
00090         memtrack_delblock_postinit(Head->Next->Ptr, NULL, 0, NULL, NULL, NULL);
00091     }
00092
00093     /* Réinitialization des pointeurs */
00094     free(Head);
00095     free(Tail);
00096     Head=NULL;
00097     Tail=NULL;
00098 }
00099
00116 static unsigned int memtrack_init() {
00117     Head = (TMemBlock *) malloc(sizeof(TMemBlock));
00118     Tail = (TMemBlock *) malloc(sizeof(TMemBlock));
00119
00120     if ((NULL==Head)|| (NULL==Tail)) {
00121         fprintf(stderr, "%s:%d Not enough memory to initialize memtracker\n",
00122             __FILE__, __LINE__);
00123         return 1;
00124     }
00125     Head->Prev = (TMemBlock *) NULL;
00126     Head->Next = Tail;
00127     Tail->Next = (TMemBlock *) NULL;
00128     Tail->Prev = Head;
00129     Tail->Ptr = Head->Ptr = (void *)NULL;
00130     Tail->Size = Head->Size = 0;

```

```

00131     Tail->File = Head->File = (char *)NULL;
00132     Tail->Line = Head->Line = 0;
00133     Tail->CompilDate = Head->CompilDate = (char *)NULL;
00134     Tail->CompilTime = Head->CompilTime = (char *)NULL;
00135     Tail->Function = Head->Function = (char *)NULL;
00136
00137     /* Initialisation des compteurs */
00138     NbBlocks=0;
00139     RAMSize=0;
00140
00141     /* Modification des foncteurs pour utiliser les fonctions définitives */
00142     memtrack_addblock = memtrack_addblock_postinit;
00143     memtrack_delblock = memtrack_delblock_postinit;
00144     memtrack_dumpblocks = memtrack_dumpblocks_postinit;
00145     memtrack_getallocatedblocks = memtrack_getallocatedblocks_postinit;
00146     memtrack_getallocatedRAM = memtrack_getallocatedRAM_postinit;
00147     memtrack_getblocksize = memtrack_getblocksize_postinit;
00148
00149     return 0;
00150 }
00151
00152
00164 static unsigned int memtrack_addblock_preinit(const void *p_Ptr,
00165     const size_t p_Size,
00166     const char *p_File,
00167     const int p_Line,
00168     const char *p_CompilDate,
00169     const char *p_CompilTime,
00170     const char *p_Function) {
00171     /* Initialisation de la liste */
00172     if (0!=memtrack_init()) {
00173         fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",
00174             __FILE__,__LINE__);
00175         /* OOM */
00176         return 1;
00177     }
00178
00179     /* Appel de la fonction réelle */
00180     return memtrack_addblock(p_Ptr, p_Size, p_File, p_Line, p_CompilDate,
00181         p_CompilTime, p_Function);
00182 }
00183
00195 static unsigned int memtrack_delblock_preinit(const void *p_Ptr,
00196     const char *p_File,
00197     const int p_Line,
00198     const char *p_CompilDate,
00199     const char *p_CompilTime,
00200     const char *p_Function) {
00201     /* Initialisation de la liste */
00202     if (0!=memtrack_init()) {
00203         fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",
00204             __FILE__,__LINE__);
00205         /* OOM */
00206         return 1;
00207     }
00208
00209     /* Appel de la fonction réelle */
00210     return memtrack_delblock(p_Ptr, p_File, p_Line, p_CompilDate,
00211         p_CompilTime, p_Function);
00212 }
00213
00225 static uint64_t memtrack_dumpblocks_preinit() {
00226     /* Initialisation de la liste */
00227     if (0!=memtrack_init()) {
00228         fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",
00229             __FILE__,__LINE__);
00230         /* OOM */
00231         return 0;
00232     }
00233
00234     /* Appel de la fonction réelle */
00235     return memtrack_dumpblocks();
00236 }
00237
00249 static uint64_t memtrack_getallocatedblocks_preinit() {
00250     /* Initialisation de la liste */
00251     if (0!=memtrack_init()) {
00252         fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",
00253             __FILE__,__LINE__);
00254         /* OOM */
00255         return 0;
00256     }
00257
00258     /* Appel de la fonction réelle */
00259     return memtrack_getallocatedblocks();
00260 }
00261

```

```

00273 static uint64_t memtrack_getallocatedRAM_preinit() {
00274     /* Initialisation de la liste */
00275     if (0!=memtrack_init()) {
00276         fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",
00277             __FILE__, __LINE__);
00278         /* OOM */
00279         return 0;
00280     }
00281
00282     /* Appel de la fonction réelle */
00283     return memtrack_getallocatedRAM();
00284 }
00285
00297 static size_t memtrack_getblocksize_preinit(const void *p_Ptr) {
00298     /* Initialisation de la liste */
00299     if (0!=memtrack_init()) {
00300         fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",
00301             __FILE__, __LINE__);
00302         /* OOM */
00303         return 0;
00304     }
00305
00306     /* Appel de la fonction réelle */
00307     return memtrack_getblocksize(p_Ptr);
00308 }
00309
00318 static unsigned int memtrack_addblock_postinit(const void *p_Ptr,
00319     const size_t p_Size,
00320     const char *p_File,
00321     const int p_Line,
00322     const char *p_CompilDate,
00323     const char *p_CompilTime,
00324     const char *p_Function) {
00325     TMemBlock *l_tmp;
00326
00327     /* Test de validité des données à enregistrer */
00328     /* Meme si malloc permet Size=0, ce n'est pas portable */
00329     if ((NULL==p_Ptr) ||
00330         (0==p_Size) ||
00331         (NULL==p_File) ||
00332         (0==p_File[0]) ||
00333         (0==p_Line) ||
00334         (NULL==p_CompilDate) ||
00335         (0==p_CompilDate[0]) ||
00336         (NULL==p_CompilTime) ||
00337         (0==p_CompilTime[0]) ||
00338         (NULL==p_Function) ||
00339         (0==p_Function[0])) {
00340         fprintf(stderr,"%s:%d Null or empty parameters\n",__FILE__, __LINE__);
00341         return 1;
00342     }
00343
00344     /* On ne peut pas dupliquer un pointeur. Pour le modifier, il faut le
00345     * supprimer et le recréer, ce n'est pas le rôle de ces fonctions de
00346     * bas-niveau */
00347
00348     /* Recherche du pointeur */
00349     l_tmp = Head->Next;
00350     while ((l_tmp->Ptr != p_Ptr) && (l_tmp != Tail))
00351         l_tmp = l_tmp->Next;
00352
00353     /* Le bloc ne doit pas avoir été trouvé */
00354     if (l_tmp != Tail) {
00355         fprintf(stderr,"%s:%d Memory bloc already registered\n",__FILE__, __LINE__);
00356         return 1;
00357     }
00358
00359     /* Allocation d'un nouveau descripteur de bloc */
00360     l_tmp = (TMemBlock *) malloc(sizeof(TMemBlock));
00361
00362     /* Allocation réussie ? */
00363     if (NULL == l_tmp) {
00364         return 1;
00365     }
00366
00367     /* Remplissage du descripteur */
00368     l_tmp->Ptr = (void *)p_Ptr;
00369     l_tmp->Size = p_Size;
00370     if (NULL==(l_tmp->File = strdup(p_File?p_File:""))) {
00371         free(l_tmp);
00372         return 1;
00373     };
00374     l_tmp->Line = p_Line;
00375     if (NULL==(l_tmp->CompilDate = strdup(p_CompilDate?p_CompilDate:""))) {
00376         free(l_tmp->File);
00377         free(l_tmp);
00378         return 1;

```



```

00379     };
00380     if (NULL==(l_tmp->CompilTime = strdup(p_CompilTime?p_CompilTime:""))) {
00381         free(l_tmp->CompilDate);
00382         free(l_tmp->File);
00383         free(l_tmp);
00384         return 1;
00385     };
00386     if (NULL==(l_tmp->Function = strdup(p_Function?p_Function:""))) {
00387         free(l_tmp->CompilTime);
00388         free(l_tmp->CompilDate);
00389         free(l_tmp->File);
00390         free(l_tmp);
00391         return 1;
00392     };
00393
00394     /* Ajout de la description dans la liste (Section critique) */
00395     l_tmp->Prev = Tail->Prev;
00396     l_tmp->Next = Tail;
00397     l_tmp->Prev->Next = l_tmp->Next->Prev = l_tmp;
00398
00399     /* Mise à jour des compteurs */
00400     NbBlocks++;
00401     RAMSize += p_Size;
00402
00403     return 0;
00404 }
00405
00414 static unsigned int memtrack_delblock_postinit(const void *p_Ptr,
00415     const char *p_File,
00416     const int p_Line,
00417     const char *p_CompilDate,
00418     const char *p_CompilTime,
00419     const char *p_Function) {
00420     TMemBlock *l_tmp;
00421     (void) p_File;
00422     (void) p_Line;
00423     (void) p_CompilDate;
00424     (void) p_CompilTime;
00425     (void) p_Function;
00426
00427     /* Recherche de la description */
00428     l_tmp = Head->Next;
00429     while ((l_tmp->Ptr != p_Ptr) && (l_tmp != Tail))
00430         l_tmp = l_tmp->Next;
00431
00432     /* Le bloc doit avoir été trouvé */
00433     if (l_tmp == Tail) {
00434         fprintf(stderr, "%s:%d Block not found for deletion\n", __FILE__, __LINE__);
00435         return 1;
00436     }
00437
00438     /* Libération des ressources acquises */
00439     /* On ne libère pas le bloc mémoire lui-même */
00440     free(l_tmp->File);
00441     free(l_tmp->CompilDate);
00442     free(l_tmp->CompilTime);
00443     free(l_tmp->Function);
00444
00445     /* Retrait de la description de la liste (Section critique) */
00446     l_tmp->Next->Prev = l_tmp->Prev;
00447     l_tmp->Prev->Next = l_tmp->Next;
00448
00449     /* Mise à jour des compteurs */
00450     NbBlocks--;
00451     RAMSize -= l_tmp->Size;
00452
00453     /* Libération de la description */
00454     free(l_tmp);
00455
00456     return 0;
00457 }
00458
00467 static uint64_t memtrack_dumpblocks_postinit() {
00468     TMemBlock *l_tmp;
00469     uint64_t l_NbBlocks = 0;;
00470
00471     if (Head->Next != Tail) {
00472         size_t l_BlockSize;
00473         fprintf(stderr,
00474
+-----+
00475             fprintf(stderr, " | %-104s |\n", "Memory Tracker Report");
00476             fprintf(stderr,
+-----+
00477
+-----+
00478             fprintf(stderr,
00479                 " | %-20s | %-20s | %-4s | %-15s | %-8s | %-22s |\n",

```


5.15 revision.h File Reference

Macros

- `#define REVISION "266ec5c3bdcc"`

5.15.1 Macro Definition Documentation

5.15.1.1 REVISION

```
#define REVISION "266ec5c3bdcc"
```

Definition at line 3 of file [revision.h](#).

5.16 revision.h

[Go to the documentation of this file.](#)

```
00001 /* This file is updated in the distdir before creating the dist archive */
00002 #ifndef REVISION
00003 #define REVISION "266ec5c3bdcc"
00004 #endif
```


Index

- [_GNU_SOURCE](#)
 - [memdbg.c, 41](#)
 - [_XOPEN_SOURCE](#)
 - [assert.c, 36](#)
 - [memdbg.c, 41](#)
 - [memtrack.c, 51](#)
 - [_trace](#)
 - [assert.c, 37](#)
 - [assert.h, 14](#)
 - [_trace_dynmsg](#)
 - [assert.c, 37](#)
 - [assert.h, 15](#)
 - [_trace_msg](#)
 - [assert.c, 38](#)
 - [assert.h, 15](#)
- [asprintf](#)
 - [memory.h, 27](#)
- [ASSERT](#)
 - [assert.h, 12](#)
- [assert.c, 35, 38](#)
 - [_XOPEN_SOURCE, 36](#)
 - [_trace, 37](#)
 - [_trace_dynmsg, 37](#)
 - [_trace_msg, 38](#)
- [assert.h, 11, 16](#)
 - [_trace, 14](#)
 - [_trace_dynmsg, 15](#)
 - [_trace_msg, 15](#)
 - [ASSERT, 12](#)
 - [DBG_ITRACE, 13](#)
 - [DBG_MSG, 13](#)
 - [DBG_PRINTF, 13](#)
 - [DBG_TRACE, 14](#)
- [calloc](#)
 - [memory.h, 27](#)
- [CompilDate](#)
 - [MemBlock, 8](#)
- [CompilTime](#)
 - [MemBlock, 8](#)
- [dbg_asprintf](#)
 - [memdbg.c, 41](#)
 - [memdbg.h, 19](#)
- [dbg_calloc](#)
 - [memdbg.c, 42](#)
 - [memdbg.h, 19](#)
- [dbg_free](#)
 - [memdbg.c, 43](#)
- [memdbg.h, 20](#)
- [DBG_ITRACE](#)
 - [assert.h, 13](#)
- [dbg_malloc](#)
 - [memdbg.c, 44](#)
 - [memdbg.h, 21](#)
- [DBG_MSG](#)
 - [assert.h, 13](#)
- [DBG_PRINTF](#)
 - [assert.h, 13](#)
- [dbg_realloc](#)
 - [memdbg.c, 45](#)
 - [memdbg.h, 22](#)
- [dbg_strdup](#)
 - [memdbg.c, 46](#)
 - [memdbg.h, 23](#)
- [DBG_TRACE](#)
 - [assert.h, 14](#)
- [File](#)
 - [MemBlock, 8](#)
- [free](#)
 - [memory.h, 27](#)
- [Function](#)
 - [MemBlock, 8](#)
- [Line](#)
 - [MemBlock, 8](#)
- [malloc](#)
 - [memory.h, 27](#)
- [MemBlock, 7](#)
 - [CompilDate, 8](#)
 - [CompilTime, 8](#)
 - [File, 8](#)
 - [Function, 8](#)
 - [Line, 8](#)
 - [Next, 9](#)
 - [Prev, 9](#)
 - [Ptr, 9](#)
 - [Size, 9](#)
- [memdbg.c, 40, 47](#)
 - [_GNU_SOURCE, 41](#)
 - [_XOPEN_SOURCE, 41](#)
 - [dbg_asprintf, 41](#)
 - [dbg_calloc, 42](#)
 - [dbg_free, 43](#)
 - [dbg_malloc, 44](#)
 - [dbg_realloc, 45](#)
 - [dbg_strdup, 46](#)

- memdbg.h, [17](#), [24](#)
 - dbg_asprintf, [19](#)
 - dbg_calloc, [19](#)
 - dbg_free, [20](#)
 - dbg_malloc, [21](#)
 - dbg_realloc, [22](#)
 - dbg_strdup, [23](#)
- memory.h, [25](#), [29](#)
 - asprintf, [27](#)
 - calloc, [27](#)
 - free, [27](#)
 - malloc, [27](#)
 - memreport, [28](#)
 - realloc, [28](#)
 - strdup, [28](#)
- memreport
 - memory.h, [28](#)
- memtrack.c, [49](#), [55](#)
 - _XOPEN_SOURCE, [51](#)
 - memtrack_addblock, [52](#)
 - memtrack_delblock, [53](#)
 - memtrack_dumpblocks, [54](#)
 - memtrack_getallocatedblocks, [54](#)
 - memtrack_getallocatedRAM, [54](#)
 - memtrack_getblocksize, [55](#)
 - memtrack_reset, [51](#)
- memtrack.h, [29](#), [34](#)
 - memtrack_addblock, [31](#)
 - memtrack_delblock, [32](#)
 - memtrack_dumpblocks, [33](#)
 - memtrack_getallocatedblocks, [33](#)
 - memtrack_getallocatedRAM, [33](#)
 - memtrack_getblocksize, [34](#)
 - TMemBlock, [31](#)
- memtrack_addblock
 - memtrack.c, [52](#)
 - memtrack.h, [31](#)
- memtrack_delblock
 - memtrack.c, [53](#)
 - memtrack.h, [32](#)
- memtrack_dumpblocks
 - memtrack.c, [54](#)
 - memtrack.h, [33](#)
- memtrack_getallocatedblocks
 - memtrack.c, [54](#)
 - memtrack.h, [33](#)
- memtrack_getallocatedRAM
 - memtrack.c, [54](#)
 - memtrack.h, [33](#)
- memtrack_getblocksize
 - memtrack.c, [55](#)
 - memtrack.h, [34](#)
- memtrack_reset
 - memtrack.c, [51](#)
- Next
 - MemBlock, [9](#)
- Prev
 - MemBlock, [9](#)
 - Ptr
 - MemBlock, [9](#)
 - realloc
 - memory.h, [28](#)
 - REVISION
 - revision.h, [61](#)
 - revision.h, [61](#)
 - REVISION, [61](#)
 - Size
 - MemBlock, [9](#)
 - strdup
 - memory.h, [28](#)
 - TMemBlock
 - memtrack.h, [31](#)