

kernel

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
4.2	mkmod_api_s Struct Reference	10
4.2.1	Detailed Description	10
4.2.2	Member Data Documentation	10
4.2.2.1	mkmod_function	10
4.3	moduleinfo_s Struct Reference	10
4.3.1	Detailed Description	11
4.3.2	Member Data Documentation	11
4.3.2.1	moduleAuthor	11
4.3.2.2	moduleDesc	11
4.3.2.3	moduleEmail	11
4.3.2.4	moduleLicense	11
4.3.2.5	moduleMajor	11
4.3.2.6	moduleMinor	12
4.3.2.7	moduleName	12
4.3.2.8	modulePatch	12
4.3.2.9	moduleURL	12
4.4	modules_s Struct Reference	12
4.4.1	Detailed Description	13
4.4.2	Member Data Documentation	13
4.4.2.1	handle	13
4.4.2.2	modinfo	13
4.4.2.3	next	13

4.5 service_s Struct Reference	14
4.5.1 Detailed Description	14
4.5.2 Member Data Documentation	14
4.5.2.1 children	14
4.5.2.2 function	14
4.5.2.3 name	15
4.5.2.4 nbArgs	15
4.5.2.5 next	15
5 File Documentation	17
5.1 ansi-color-codes.h File Reference	17
5.1.1 Macro Definition Documentation	18
5.1.1.1 BBLK	19
5.1.1.2 BBLU	19
5.1.1.3 BCYN	19
5.1.1.4 BGRN	19
5.1.1.5 BHBLK	19
5.1.1.6 BHBLU	19
5.1.1.7 BHCCYN	20
5.1.1.8 BHGRN	20
5.1.1.9 BHMAG	20
5.1.1.10 BHRED	20
5.1.1.11 BHWHT	20
5.1.1.12 BHYEL	20
5.1.1.13 BLINK	21
5.1.1.14 BLK	21
5.1.1.15 BLKB	21
5.1.1.16 BLKHB	21
5.1.1.17 BLU	21
5.1.1.18 BLUB	21
5.1.1.19 BLUHB	22
5.1.1.20 BMAG	22
5.1.1.21 BOLD	22
5.1.1.22 BRED	22
5.1.1.23 BWHT	22
5.1.1.24 BYEL	22
5.1.1.25 CYN	23
5.1.1.26 CYNB	23
5.1.1.27 CYNHB	23
5.1.1.28 DIM	23
5.1.1.29 GRN	23
5.1.1.30 GRNB	23

5.1.1.31 GRNHB	24
5.1.1.32 HBLK	24
5.1.1.33 HBLU	24
5.1.1.34 HCYN	24
5.1.1.35 HGRN	24
5.1.1.36 HIDDEN	24
5.1.1.37 HMAG	25
5.1.1.38 HRED	25
5.1.1.39 HWHT	25
5.1.1.40 HYEL	25
5.1.1.41 MAG	25
5.1.1.42 MAGB	25
5.1.1.43 MAGHB	26
5.1.1.44 RED	26
5.1.1.45 REDB	26
5.1.1.46 REDHB	26
5.1.1.47 RESET	26
5.1.1.48 REVERSE	26
5.1.1.49 STRIKE	27
5.1.1.50 UBLK	27
5.1.1.51 UBLU	27
5.1.1.52 UCYN	27
5.1.1.53 UGRN	27
5.1.1.54 UMAG	27
5.1.1.55 UNDERLINE	28
5.1.1.56 URED	28
5.1.1.57 UWHT	28
5.1.1.58 UYEL	28
5.1.1.59 WHT	28
5.1.1.60 WHTB	28
5.1.1.61 WHTHB	29
5.1.1.62 YEL	29
5.1.1.63 YELB	29
5.1.1.64 YELHB	29
5.2 ansi-color-codes.h	29
5.3 assert.c File Reference	30
5.3.1 Detailed Description	31
5.3.2 Function Documentation	32
5.3.2.1 _trace()	32
5.3.2.2 _trace_dynmsg()	32
5.3.2.3 _trace_msg()	33
5.4 assert.c	33

5.5 assert.h File Reference	35
5.5.1 Detailed Description	36
5.5.2 Macro Definition Documentation	36
5.5.2.1 ASSERT	36
5.5.2.2 DBG_ITRACE	37
5.5.2.3 DBG_MSG	37
5.5.2.4 DBG_PRINTF	37
5.5.2.5 DBG_TRACE	38
5.5.3 Function Documentation	38
5.5.3.1 _trace()	38
5.5.3.2 _trace_dynmsg()	38
5.5.3.3 _trace_msg()	39
5.6 assert.h	39
5.7 memdbg.c File Reference	41
5.7.1 Detailed Description	42
5.7.2 Function Documentation	42
5.7.2.1 dbg_asprintf()	42
5.7.2.2 dbg_calloc()	43
5.7.2.3 dbg_free()	44
5.7.2.4 dbg_malloc()	45
5.7.2.5 dbg_realloc()	46
5.7.2.6 dbg_strdup()	47
5.8 memdbg.c	48
5.9 memdbg.h File Reference	50
5.9.1 Detailed Description	52
5.9.2 Function Documentation	52
5.9.2.1 dbg_asprintf()	52
5.9.2.2 dbg_calloc()	53
5.9.2.3 dbg_free()	54
5.9.2.4 dbg_malloc()	55
5.9.2.5 dbg_realloc()	56
5.9.2.6 dbg_strdup()	57
5.10 memdbg.h	58
5.11 memory.h File Reference	59
5.11.1 Detailed Description	60
5.11.2 Macro Definition Documentation	60
5.11.2.1 asprintf	61
5.11.2.2 calloc	61
5.11.2.3 free	61
5.11.2.4 malloc	61
5.11.2.5 memreport	62
5.11.2.6 realloc	62

5.11.2.7 strdup	62
5.12 memory.h	62
5.13 memtrack.c File Reference	63
5.13.1 Detailed Description	64
5.13.2 Function Documentation	64
5.13.2.1 memtrack_reset()	64
5.13.3 Variable Documentation	64
5.13.3.1 memtrack_addblock	65
5.13.3.2 memtrack_delblock	66
5.13.3.3 memtrack_dumpblocks	66
5.13.3.4 memtrack_getallocatedblocks	67
5.13.3.5 memtrack_getallocatedRAM	67
5.13.3.6 memtrack_getblocksize	68
5.14 memtrack.c	68
5.15 memtrack.h File Reference	74
5.15.1 Detailed Description	75
5.15.2 Typedef Documentation	75
5.15.2.1 TMemBlock	75
5.15.3 Variable Documentation	76
5.15.3.1 memtrack_addblock	76
5.15.3.2 memtrack_delblock	77
5.15.3.3 memtrack_dumpblocks	77
5.15.3.4 memtrack_getallocatedblocks	78
5.15.3.5 memtrack_getallocatedRAM	78
5.15.3.6 memtrack_getblocksize	79
5.16 memtrack.h	79
5.17 oom.c File Reference	80
5.17.1 Detailed Description	81
5.17.2 Macro Definition Documentation	81
5.17.2.1 __asm__	81
5.17.2.2 __sync_synchronize	81
5.17.2.3 _GNU_SOURCE	81
5.17.2.4 RAMBLOCKS_MAX	82
5.17.3 Function Documentation	82
5.17.3.1 oomtest_config()	82
5.17.3.2 oomtest_enabled()	83
5.17.4 Variable Documentation	83
5.17.4.1 oomtest_disable	83
5.17.4.2 oomtest_enable	83
5.17.4.3 oomtest_fill	84
5.17.4.4 oomtest_free	85
5.18 oom.c	85

5.19 oom.h File Reference	89
5.19.1 Detailed Description	90
5.19.2 Macro Definition Documentation	90
5.19.2.1 RAMLIMIT_HARD	91
5.19.2.2 RAMLIMIT_SOFT	91
5.19.3 Function Documentation	91
5.19.3.1 oomtest_config()	91
5.19.3.2 oomtest_enabled()	92
5.19.4 Variable Documentation	92
5.19.4.1 oomtest_disable	92
5.19.4.2 oomtest_enable	93
5.19.4.3 oomtest_fill	93
5.19.4.4 oomtest_free	94
5.20 oom.h	94
5.21 gettext.h File Reference	95
5.21.1 Macro Definition Documentation	96
5.21.1.1 _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS	97
5.21.1.2 bind_textdomain_codeset	97
5.21.1.3 bindtextdomain	97
5.21.1.4 dcgettext	97
5.21.1.5 dcngettext	97
5.21.1.6 dcnpgettext	98
5.21.1.7 dcpgettext	98
5.21.1.8 dgettext	98
5.21.1.9 dngettext	98
5.21.1.10 dnpgettext	99
5.21.1.11 dnpgettext_expr	99
5.21.1.12 dpgettext	99
5.21.1.13 dpgettext_expr	99
5.21.1.14 gettext	100
5.21.1.15 GETTEXT_CONTEXT_GLUE	100
5.21.1.16 gettext_noop	100
5.21.1.17 ngettext	100
5.21.1.18 npgettext	100
5.21.1.19 npgettext_expr	101
5.21.1.20 pgettext	101
5.21.1.21 pgettext_expr	101
5.21.1.22 textdomain	101
5.22 gettext.h	102
5.23 mkernel-opt.c File Reference	105
5.23.1 Macro Definition Documentation	107
5.23.1.1 HELP_DESC	107

5.23.1.2 HELP_name	107
5.23.1.3 LOAD_OPTS_DESC	107
5.23.1.4 LOAD_OPTS_NAME	108
5.23.1.5 LOAD_OPTS_name	108
5.23.1.6 LOAD_OPTS_pfx	108
5.23.1.7 mkernel_full_usage	108
5.23.1.8 mkernel_packager_info	108
5.23.1.9 mkernel_short_usage	108
5.23.1.10 MODULE_PATH_DESC	109
5.23.1.11 MODULE_PATH_FLAGS	109
5.23.1.12 MODULE_PATH_NAME	109
5.23.1.13 MODULE_PATH_name	109
5.23.1.14 MORE_HELP_DESC	109
5.23.1.15 MORE_HELP_FLAGS	110
5.23.1.16 MORE_HELP_name	110
5.23.1.17 NO_LOAD_OPTS_name	110
5.23.1.18 NULL	110
5.23.1.19 OPTION_CODE_COMPILE	110
5.23.1.20 OPTPROC_BASE	110
5.23.1.21 PKGDATADIR	111
5.23.1.22 SAVE_OPTS_DESC	111
5.23.1.23 SAVE_OPTS_name	111
5.23.1.24 translate_option_strings	111
5.23.1.25 VER_DESC	111
5.23.1.26 VER_FLAGS	111
5.23.1.27 VER_name	112
5.23.1.28 VER_PROC	112
5.23.1.29 zBugsAddr	112
5.23.1.30 zCopyright	112
5.23.1.31 zDetail	112
5.23.1.32 zExplain	112
5.23.1.33 zFullVersion	113
5.23.1.34 zLicenseDescrip	113
5.23.1.35 zPROGNAME	113
5.23.1.36 zRcName	113
5.23.1.37 zUsageTitle	113
5.23.2 Variable Documentation	113
5.23.2.1 mkernelOptions	114
5.23.2.2 option_usage_fp	114
5.23.2.3 optionBooleanVal	114
5.23.2.4 optionNestedVal	114
5.23.2.5 optionNumericVal	114

5.23.2.6 optionPagedUsage	114
5.23.2.7 optionPrintVersion	115
5.23.2.8 optionResetOpt	115
5.23.2.9 optionStackArg	115
5.23.2.10 optionTimeDate	115
5.23.2.11 optionTimeVal	115
5.23.2.12 optionUnstackArg	115
5.23.2.13 optionVendorOption	116
5.24 mkkernel-opt.c	116
5.25 mkkernel.c File Reference	125
5.25.1 Detailed Description	126
5.25.2 Macro Definition Documentation	126
5.25.2.1 _	126
5.25.2.2 MODULE_PATH_DEFAULT	126
5.25.2.3 MODULE_PATH_ENV	127
5.25.2.4 PATH_MAX	127
5.25.3 Function Documentation	127
5.25.3.1 main()	127
5.26 mkkernel.c	128
5.27 mkmod.h File Reference	130
5.27.1 Detailed Description	131
5.27.2 Typedef Documentation	131
5.27.2.1 mkmod_api_t	131
5.28 mkmod.h	131
5.29 mkmodgtk.c File Reference	131
5.29.1 Detailed Description	132
5.29.2 Macro Definition Documentation	133
5.29.2.1 _	133
5.29.3 Function Documentation	133
5.29.3.1 onLoad()	133
5.29.3.2 onUnload()	133
5.29.4 Variable Documentation	133
5.29.4.1 module_api	133
5.30 mkmodgtk.c	134
5.31 mkmodtty.c File Reference	135
5.31.1 Detailed Description	136
5.31.2 Macro Definition Documentation	136
5.31.2.1 _	136
5.31.3 Function Documentation	136
5.31.3.1 onLoad()	136
5.31.3.2 onUnload()	137
5.31.4 Variable Documentation	137

5.31.4.1 module_api	137
5.32 mkmodtty.c	137
5.33 modmgr.c File Reference	138
5.33.1 Detailed Description	139
5.33.2 Macro Definition Documentation	139
5.33.2.1 _	139
5.33.2.2 PATH_MAX	140
5.33.3 Typedef Documentation	140
5.33.3.1 modules_t	140
5.33.4 Function Documentation	140
5.33.4.1 modmgr_addpath()	140
5.33.4.2 modmgr_getpath()	141
5.33.4.3 modmgr_getsymbol()	141
5.33.4.4 modmgr_insertpath()	141
5.33.4.5 modmgr_list()	141
5.33.4.6 modmgr_load()	142
5.33.4.7 modmgr_setpath()	142
5.33.4.8 modmgr_unload()	142
5.34 modmgr.c	143
5.35 modmgr.h File Reference	148
5.35.1 Detailed Description	149
5.35.2 Macro Definition Documentation	149
5.35.2.1 MODMGR_GETFUNCTION	149
5.35.2.2 MODMGR_LOAD	150
5.35.3 Typedef Documentation	150
5.35.3.1 modmgr_module_t	150
5.35.4 Function Documentation	150
5.35.4.1 modmgr_addpath()	150
5.35.4.2 modmgr_getpath()	151
5.35.4.3 modmgr_getsymbol()	151
5.35.4.4 modmgr_insertpath()	151
5.35.4.5 modmgr_list()	151
5.35.4.6 modmgr_load()	152
5.35.4.7 modmgr_setpath()	152
5.35.4.8 modmgr_unload()	152
5.36 modmgr.h	153
5.37 module.h File Reference	153
5.37.1 Detailed Description	154
5.37.2 Typedef Documentation	154
5.37.2.1 moduleinfo_t	155
5.38 module.h	155
5.39 revision.h File Reference	155

5.39.1 Macro Definition Documentation	156
5.39.1.1 REVISION	156
5.40 revision.h	156
5.41 svcmgr.c File Reference	156
5.41.1 Detailed Description	157
5.41.2 Macro Definition Documentation	157
5.41.2.1 _	157
5.41.3 Typedef Documentation	158
5.41.3.1 service_t	158
5.41.4 Function Documentation	158
5.41.4.1 svcmgr_call()	158
5.41.4.2 svcmgr_dump()	158
5.41.4.3 svcmgr_register()	159
5.41.4.4 svcmgr_unregister()	159
5.42 svcmgr.c	159
5.43 svcmgr.h File Reference	162
5.43.1 Detailed Description	163
5.43.2 Typedef Documentation	163
5.43.2.1 svcfunc_t	163
5.43.3 Function Documentation	163
5.43.3.1 svcmgr_call()	163
5.43.3.2 svcmgr_register()	164
5.43.3.3 svcmgr_unregister()	164
5.44 svcmgr.h	164
Index	165

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

Member `modmgr_load` (`const char *modfile`)

critical section

Member `MODULE_PATH_ENV`

Define in `configure.ac` with default value

File `svcmgr.c`

make threadsafe

investigate prefix or `b+*` trees

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
mkmod_api_s	10
moduleinfo_s	10
modules_s		
	Module list item structure	12
service_s	14

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

ansi-color-codes.h	17
assert.c	
Compiled functions used by debugging macros to write on stderr	30
assert.h	
Debugging macros	35
memdbg.c	
Memory leak tracker implementation	41
memdbg.h	
Memory leak tracker header	50
memory.h	
Tracks memory allocation and leaks when compiled without NDEBUG	59
memtrack.c	
Memory block metadata tracking implementation	63
memtrack.h	
Memory block metadata tracking headers	74
oom.c	80
oom.h	89
gettext.h	95
mkernel-opt.c	105
mkernel.c	
Micro-kernel core main source	125
mkmod.h	
ABI interface shared between module class and application	130
mkmodgtk.c	131
mkmodtty.c	135
modmgr.c	
Module manager implementation	138
modmgr.h	
Module manager headers	148
module.h	
Internal ABI shared by all modules with modmgr	153
revision.h	155
svcmgr.c	
Service manager implementation	156
svcmgr.h	
Service manager header	162

Chapter 4

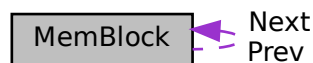
Class Documentation

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 [37](#) 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 [44](#) of file [memtrack.h](#).

4.1.2.2 `CompilTime`

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

Source file compilation time.

Definition at line [45](#) of file [memtrack.h](#).

4.1.2.3 `File`

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

Source file which asked the allocation.

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

4.1.2.4 `Function`

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

Function name which asked the allocation.

Definition at line [46](#) of file [memtrack.h](#).

4.1.2.5 Line

```
int MemBlock::Line
```

Source line number ch asked the allocation.

Definition at line [43](#) of file [memtrack.h](#).

4.1.2.6 Next

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

Next item pointer.

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

4.1.2.7 Prev

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

Previous item pointer.

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

4.1.2.8 Ptr

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

Allocated memory block pointer.

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

4.1.2.9 Size

```
size_t MemBlock::Size
```

Allocated memory block size.

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

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

- [memtrack.h](#)

4.2 mkmod_api_s Struct Reference

```
#include <mkmod.h>
```

Public Attributes

- void(* [mkmod_function](#))()

4.2.1 Detailed Description

Definition at line 24 of file [mkmod.h](#).

4.2.2 Member Data Documentation

4.2.2.1 mkmod_function

```
void(* mkmod_api_s::mkmod_function) ()
```

Definition at line 25 of file [mkmod.h](#).

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

- [mkmod.h](#)

4.3 moduleinfo_s Struct Reference

```
#include <module.h>
```

Public Attributes

- const char * [moduleName](#)
- const char * [moduleDesc](#)
- const uint8_t [moduleMajor](#)
- const uint8_t [moduleMinor](#)
- const uint8_t [modulePatch](#)
- const char * [moduleAuthor](#)
- const char * [moduleEmail](#)
- const char * [moduleURL](#)
- const char * [moduleLicense](#)

4.3.1 Detailed Description

Definition at line 27 of file [module.h](#).

4.3.2 Member Data Documentation

4.3.2.1 moduleAuthor

```
const char* moduleinfo_s::moduleAuthor
```

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

4.3.2.2 moduleDesc

```
const char* moduleinfo_s::moduleDesc
```

Definition at line 29 of file [module.h](#).

4.3.2.3 moduleEmail

```
const char* moduleinfo_s::moduleEmail
```

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

4.3.2.4 moduleLicense

```
const char* moduleinfo_s::moduleLicense
```

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

4.3.2.5 moduleMajor

```
const uint8_t moduleinfo_s::moduleMajor
```

Definition at line 30 of file [module.h](#).

4.3.2.6 moduleMinor

```
const uint8_t moduleinfo_s::moduleMinor
```

Definition at line 31 of file [module.h](#).

4.3.2.7 moduleName

```
const char* moduleinfo_s::moduleName
```

Definition at line 28 of file [module.h](#).

4.3.2.8 modulePatch

```
const uint8_t moduleinfo_s::modulePatch
```

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

4.3.2.9 moduleURL

```
const char* moduleinfo_s::moduleURL
```

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

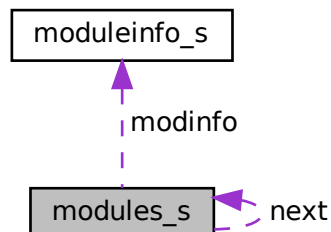
The documentation for this struct was generated from the following file:

- [module.h](#)

4.4 modules_s Struct Reference

Module list item structure.

Collaboration diagram for `modules_s`:



Public Attributes

- `lt_dlhandle` [handle](#)
- `moduleinfo_t*` [modinfo](#)
- `struct modules_s*` [next](#)

4.4.1 Detailed Description

Module list item structure.

Definition at line 42 of file [modmgr.c](#).

4.4.2 Member Data Documentation

4.4.2.1 handle

```
lt_dlhandle modules_s::handle
```

Definition at line 43 of file [modmgr.c](#).

4.4.2.2 modinfo

```
moduleinfo_t* modules_s::modinfo
```

Definition at line 44 of file [modmgr.c](#).

4.4.2.3 next

```
struct modules_s* modules_s::next
```

Definition at line 45 of file [modmgr.c](#).

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

- [modmgr.c](#)

4.5 service_s Struct Reference

Collaboration diagram for service_s:



Public Attributes

- struct [service_s](#) * [next](#)
- struct [service_s](#) * [children](#)
- uint8_t [nbArgs](#)
- char * [name](#)
- [svcfunc_t](#) * [function](#)

4.5.1 Detailed Description

Definition at line 35 of file [svcmgr.c](#).

4.5.2 Member Data Documentation

4.5.2.1 children

```
struct service\_s* service\_s::children
```

Definition at line 37 of file [svcmgr.c](#).

4.5.2.2 function

```
svcfunc\_t* service\_s::function
```

Definition at line 40 of file [svcmgr.c](#).

4.5.2.3 `name`

```
char* service_s::name
```

Definition at line 39 of file [svcmgr.c](#).

4.5.2.4 `nbArgs`

```
uint8_t service_s::nbArgs
```

Definition at line 38 of file [svcmgr.c](#).

4.5.2.5 `next`

```
struct service\_s* service_s::next
```

Definition at line 36 of file [svcmgr.c](#).

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

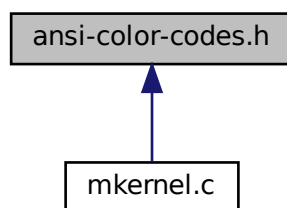
- [svcmgr.c](#)

Chapter 5

File Documentation

5.1 ansi-color-codes.h File Reference

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



Macros

- #define **BLK** "\33[0;30m"
- #define **RED** "\33[0;31m"
- #define **GRN** "\33[0;32m"
- #define **YEL** "\33[0;33m"
- #define **BLU** "\33[0;34m"
- #define **MAG** "\33[0;35m"
- #define **CYN** "\33[0;36m"
- #define **WHT** "\33[0;37m"
- #define **BBLK** "\33[1;30m"
- #define **BRED** "\33[1;31m"
- #define **BGRN** "\33[1;32m"
- #define **BYEL** "\33[1;33m"
- #define **BBLU** "\33[1;34m"
- #define **BMAG** "\33[1;35m"
- #define **BCYN** "\33[1;36m"
- #define **BWHT** "\33[1;37m"

- #define UBLK "\33[4;30m"
- #define URED "\33[4;31m"
- #define UGRN "\33[4;32m"
- #define UYEL "\33[4;33m"
- #define UBLU "\33[4;34m"
- #define UMAG "\33[4;35m"
- #define UCYN "\33[4;36m"
- #define UWHT "\33[4;37m"
- #define BLKB "\33[40m"
- #define REDB "\33[41m"
- #define GRNB "\33[42m"
- #define YELB "\33[43m"
- #define BLUB "\33[44m"
- #define MAGB "\33[45m"
- #define CYNB "\33[46m"
- #define WHTB "\33[47m"
- #define BLKHB "\33[0;100m"
- #define REDHB "\33[0;101m"
- #define GRNHB "\33[0;102m"
- #define YELHB "\33[0;103m"
- #define BLUHB "\33[0;104m"
- #define MAGHB "\33[0;105m"
- #define CYNHB "\33[0;106m"
- #define WHTHB "\33[0;107m"
- #define HBLK "\33[0;90m"
- #define HRED "\33[0;91m"
- #define HGRN "\33[0;92m"
- #define HYEL "\33[0;93m"
- #define HBLU "\33[0;94m"
- #define HMAG "\33[0;95m"
- #define HCYN "\33[0;96m"
- #define HWHT "\33[0;97m"
- #define BHBLK "\33[1;90m"
- #define BHRED "\33[1;91m"
- #define BHGRN "\33[1;92m"
- #define BHYEL "\33[1;93m"
- #define BHBLU "\33[1;94m"
- #define BHMAG "\33[1;95m"
- #define BHCYN "\33[1;96m"
- #define BHWHT "\33[1;97m"
- #define RESET "\33[0m"
- #define DIM "\33[22m"
- #define BLINK "\33[5m"
- #define HIDDEN "\33[8m"
- #define REVERSE "\33[7m"
- #define BOLD "\33[1m"
- #define UNDERLINE "\33[4m"
- #define STRIKE "\33[9m"

5.1.1 Macro Definition Documentation

5.1.1.1 BBLK

```
#define BBLK "\33[1;30m"
```

Definition at line 20 of file [ansi-color-codes.h](#).

5.1.1.2 BBLU

```
#define BBLU "\33[1;34m"
```

Definition at line 24 of file [ansi-color-codes.h](#).

5.1.1.3 BCYN

```
#define BCYN "\33[1;36m"
```

Definition at line 26 of file [ansi-color-codes.h](#).

5.1.1.4 BGRN

```
#define BGRN "\33[1;32m"
```

Definition at line 22 of file [ansi-color-codes.h](#).

5.1.1.5 BHBLK

```
#define BHBLK "\33[1;90m"
```

Definition at line 70 of file [ansi-color-codes.h](#).

5.1.1.6 BHBLU

```
#define BHBLU "\33[1;94m"
```

Definition at line 74 of file [ansi-color-codes.h](#).

5.1.1.7 BHCYN

```
#define BHCYN "\33[1;96m"
```

Definition at line 76 of file [ansi-color-codes.h](#).

5.1.1.8 BHGRN

```
#define BHGRN "\33[1;92m"
```

Definition at line 72 of file [ansi-color-codes.h](#).

5.1.1.9 BHMAG

```
#define BHMAG "\33[1;95m"
```

Definition at line 75 of file [ansi-color-codes.h](#).

5.1.1.10 BHRED

```
#define BHRED "\33[1;91m"
```

Definition at line 71 of file [ansi-color-codes.h](#).

5.1.1.11 BHWHT

```
#define BHWHT "\33[1;97m"
```

Definition at line 77 of file [ansi-color-codes.h](#).

5.1.1.12 BHYEL

```
#define BHYEL "\33[1;93m"
```

Definition at line 73 of file [ansi-color-codes.h](#).

5.1.1.13 BLINK

```
#define BLINK "\33[5m"
```

Definition at line 82 of file [ansi-color-codes.h](#).

5.1.1.14 BLK

```
#define BLK "\33[0;30m"
```

Definition at line 10 of file [ansi-color-codes.h](#).

5.1.1.15 BLKB

```
#define BLKB "\33[40m"
```

Definition at line 40 of file [ansi-color-codes.h](#).

5.1.1.16 BLKHB

```
#define BLKHB "\33[0;100m"
```

Definition at line 50 of file [ansi-color-codes.h](#).

5.1.1.17 BLU

```
#define BLU "\33[0;34m"
```

Definition at line 14 of file [ansi-color-codes.h](#).

5.1.1.18 BLUB

```
#define BLUB "\33[44m"
```

Definition at line 44 of file [ansi-color-codes.h](#).

5.1.1.19 **BLUHB**

```
#define BLUHB "\33[0;104m"
```

Definition at line 54 of file [ansi-color-codes.h](#).

5.1.1.20 **BMAG**

```
#define BMAG "\33[1;35m"
```

Definition at line 25 of file [ansi-color-codes.h](#).

5.1.1.21 **BOLD**

```
#define BOLD "\33[1m"
```

Definition at line 85 of file [ansi-color-codes.h](#).

5.1.1.22 **BRED**

```
#define BRED "\33[1;31m"
```

Definition at line 21 of file [ansi-color-codes.h](#).

5.1.1.23 **BWHT**

```
#define BWHT "\33[1;37m"
```

Definition at line 27 of file [ansi-color-codes.h](#).

5.1.1.24 **BYEL**

```
#define BYEL "\33[1;33m"
```

Definition at line 23 of file [ansi-color-codes.h](#).

5.1.1.25 CYN

```
#define CYN "\33[0;36m"
```

Definition at line 16 of file [ansi-color-codes.h](#).

5.1.1.26 CYNB

```
#define CYNB "\33[46m"
```

Definition at line 46 of file [ansi-color-codes.h](#).

5.1.1.27 CYNHB

```
#define CYNHB "\33[0;106m"
```

Definition at line 56 of file [ansi-color-codes.h](#).

5.1.1.28 DIM

```
#define DIM "\33[22m"
```

Definition at line 81 of file [ansi-color-codes.h](#).

5.1.1.29 GRN

```
#define GRN "\33[0;32m"
```

Definition at line 12 of file [ansi-color-codes.h](#).

5.1.1.30 GRNB

```
#define GRNB "\33[42m"
```

Definition at line 42 of file [ansi-color-codes.h](#).

5.1.1.31 GRNHB

```
#define GRNHB "\33[0;102m"
```

Definition at line 52 of file [ansi-color-codes.h](#).

5.1.1.32 HBLK

```
#define HBLK "\33[0;90m"
```

Definition at line 60 of file [ansi-color-codes.h](#).

5.1.1.33 HBLU

```
#define HBLU "\33[0;94m"
```

Definition at line 64 of file [ansi-color-codes.h](#).

5.1.1.34 HCYN

```
#define HCYN "\33[0;96m"
```

Definition at line 66 of file [ansi-color-codes.h](#).

5.1.1.35 HGRN

```
#define HGRN "\33[0;92m"
```

Definition at line 62 of file [ansi-color-codes.h](#).

5.1.1.36 HIDDEN

```
#define HIDDEN "\33[8m"
```

Definition at line 83 of file [ansi-color-codes.h](#).

5.1.1.37 HMAG

```
#define HMAG "\33[0;95m"
```

Definition at line 65 of file [ansi-color-codes.h](#).

5.1.1.38 HRED

```
#define HRED "\33[0;91m"
```

Definition at line 61 of file [ansi-color-codes.h](#).

5.1.1.39 HWHT

```
#define HWHT "\33[0;97m"
```

Definition at line 67 of file [ansi-color-codes.h](#).

5.1.1.40 HYEL

```
#define HYEL "\33[0;93m"
```

Definition at line 63 of file [ansi-color-codes.h](#).

5.1.1.41 MAG

```
#define MAG "\33[0;35m"
```

Definition at line 15 of file [ansi-color-codes.h](#).

5.1.1.42 MAGB

```
#define MAGB "\33[45m"
```

Definition at line 45 of file [ansi-color-codes.h](#).

5.1.1.43 MAGHB

```
#define MAGHB "\33[0;105m"
```

Definition at line 55 of file [ansi-color-codes.h](#).

5.1.1.44 RED

```
#define RED "\33[0;31m"
```

Definition at line 11 of file [ansi-color-codes.h](#).

5.1.1.45 REDB

```
#define REDB "\33[41m"
```

Definition at line 41 of file [ansi-color-codes.h](#).

5.1.1.46 REDHB

```
#define REDHB "\33[0;101m"
```

Definition at line 51 of file [ansi-color-codes.h](#).

5.1.1.47 RESET

```
#define RESET "\33[0m"
```

Definition at line 80 of file [ansi-color-codes.h](#).

5.1.1.48 REVERSE

```
#define REVERSE "\33[7m"
```

Definition at line 84 of file [ansi-color-codes.h](#).

5.1.1.49 STRIKE

```
#define STRIKE "\33[9m"
```

Definition at line 87 of file [ansi-color-codes.h](#).

5.1.1.50 UBLK

```
#define UBLK "\33[4;30m"
```

Definition at line 30 of file [ansi-color-codes.h](#).

5.1.1.51 UBLU

```
#define UBLU "\33[4;34m"
```

Definition at line 34 of file [ansi-color-codes.h](#).

5.1.1.52 UCYN

```
#define UCYN "\33[4;36m"
```

Definition at line 36 of file [ansi-color-codes.h](#).

5.1.1.53 UGRN

```
#define UGRN "\33[4;32m"
```

Definition at line 32 of file [ansi-color-codes.h](#).

5.1.1.54 UMAG

```
#define UMAG "\33[4;35m"
```

Definition at line 35 of file [ansi-color-codes.h](#).

5.1.1.55 UNDERLINE

```
#define UNDERLINE "\33[4m"
```

Definition at line 86 of file [ansi-color-codes.h](#).

5.1.1.56 URED

```
#define URED "\33[4;31m"
```

Definition at line 31 of file [ansi-color-codes.h](#).

5.1.1.57 UWHT

```
#define UWHT "\33[4;37m"
```

Definition at line 37 of file [ansi-color-codes.h](#).

5.1.1.58 UYEL

```
#define UYEL "\33[4;33m"
```

Definition at line 33 of file [ansi-color-codes.h](#).

5.1.1.59 WHT

```
#define WHT "\33[0;37m"
```

Definition at line 17 of file [ansi-color-codes.h](#).

5.1.1.60 WHTB

```
#define WHTB "\33[47m"
```

Definition at line 47 of file [ansi-color-codes.h](#).

5.1.1.61 WHTHB

```
#define WHTHB "\33[0;107m"
```

Definition at line 57 of file [ansi-color-codes.h](#).

5.1.1.62 YEL

```
#define YEL "\33[0;33m"
```

Definition at line 13 of file [ansi-color-codes.h](#).

5.1.1.63 YELB

```
#define YELB "\33[43m"
```

Definition at line 43 of file [ansi-color-codes.h](#).

5.1.1.64 YELHB

```
#define YELHB "\33[0;103m"
```

Definition at line 53 of file [ansi-color-codes.h](#).

5.2 ansi-color-codes.h

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

```
00001 /*
00002 * This is free and unencumbered software released into the public domain.
00003 *
00004 * For more information, please refer to <https://unlicense.org>
00005 *
00006 * Downloaded from https://gist.github.com/federicheddu/036ddc1624c12c073d1d481f3044628a
00007 */
00008
00009 //Regular text
00010 #define BLK "\33[0;30m"
00011 #define RED "\33[0;31m"
00012 #define GRN "\33[0;32m"
00013 #define YEL "\33[0;33m"
00014 #define BLU "\33[0;34m"
00015 #define MAG "\33[0;35m"
00016 #define CYN "\33[0;36m"
00017 #define WHT "\33[0;37m"
00018
00019 //Regular bold text
00020 #define BBLK "\33[1;30m"
00021 #define BRED "\33[1;31m"
00022 #define BGRN "\33[1;32m"
00023 #define BYEL "\33[1;33m"
00024 #define BBLU "\33[1;34m"
00025 #define BMAG "\33[1;35m"
00026 #define BCYN "\33[1;36m"
```

```

00027 #define BWHT "\33[1;37m"
00028
00029 //Regular underline text
00030 #define UBLK "\33[4;30m"
00031 #define URED "\33[4;31m"
00032 #define UGRN "\33[4;32m"
00033 #define UYEL "\33[4;33m"
00034 #define UBLU "\33[4;34m"
00035 #define UMAG "\33[4;35m"
00036 #define UCYN "\33[4;36m"
00037 #define UWHT "\33[4;37m"
00038
00039 //Regular background
00040 #define BLKB "\33[40m"
00041 #define REDB "\33[41m"
00042 #define GRNB "\33[42m"
00043 #define YELB "\33[43m"
00044 #define BLUB "\33[44m"
00045 #define MAGB "\33[45m"
00046 #define CYNB "\33[46m"
00047 #define WHTB "\33[47m"
00048
00049 //High intensty background
00050 #define BLKHB "\33[0;100m"
00051 #define REDHB "\33[0;101m"
00052 #define GRNHB "\33[0;102m"
00053 #define YELHB "\33[0;103m"
00054 #define BLUHB "\33[0;104m"
00055 #define MAGHB "\33[0;105m"
00056 #define CYNHB "\33[0;106m"
00057 #define WHTHB "\33[0;107m"
00058
00059 //High intensty text
00060 #define HBLK "\33[0;90m"
00061 #define HRED "\33[0;91m"
00062 #define HGRN "\33[0;92m"
00063 #define HYEL "\33[0;93m"
00064 #define HBLU "\33[0;94m"
00065 #define HMAG "\33[0;95m"
00066 #define HCYN "\33[0;96m"
00067 #define HWHT "\33[0;97m"
00068
00069 //Bold high intensity text
00070 #define BHBLK "\33[1;90m"
00071 #define BHRED "\33[1;91m"
00072 #define BHGRN "\33[1;92m"
00073 #define BHYEL "\33[1;93m"
00074 #define BHBLU "\33[1;94m"
00075 #define BHMAG "\33[1;95m"
00076 #define BHCYN "\33[1;96m"
00077 #define BHWHT "\33[1;97m"
00078
00079 //Reset
00080 #define RESET "\33[0m"
00081 #define DIM "\33[22m"
00082 #define BLINK "\33[5m"
00083 #define HIDDEN "\33[8m"
00084 #define REVERSE "\33[7m"
00085 #define BOLD "\33[1m"
00086 #define UNDERLINE "\33[4m"
00087 #define STRIKE "\33[9m"

```

5.3 assert.c File Reference

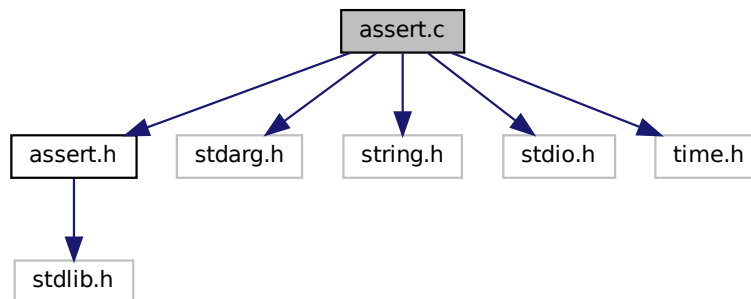
Compiled functions used by debugging macros to write on stderr.

```

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

```

Include dependency graph for assert.c:



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.

5.3.1 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](#).

5.3.2 Function Documentation

5.3.2.1 `_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 75 of file [assert.c](#).

5.3.2.2 `_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 sprintf function

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

5.3.2.3 _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 87 of file [assert.c](#).

5.4 assert.c

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

```
00001
00020 #include "assert.h"
00021 #include <stdarg.h> /* va_list, va_start, va_arg, va_end */
00022 #include <string.h> /* strcpy */
00023 #include <stdio.h> /* fprintf */
00024 #include <time.h> /* time, localtime */
00025
00041 static const char*_timestamp(const char* p_File,
00042                             const unsigned int p_Line,
00043                             const char* p_CompilDate,
00044                             const char* p_CompilTime,
00045                             const char* p_Function)
00046 {
00047     /* Get local time and format it */
00048     char l_Time[24];
00049     static char l_tmp[110];
00050     time_t l_CurrentTime = time(NULL);
00051
```

```

00052     /* Parameter validity check against invalid parameters such as NULL or
00053  * empty string values. */
00054     if ((NULL==p_File)|| (0==p_File[0])||
00055         (0==p_Line)||
00056         (NULL==p_CompilDate)|| (0==p_CompilDate[0])||
00057         (NULL==p_CompilTime)|| (0==p_CompilTime[0])||
00058         (NULL==p_Function)|| (0==p_Function[0])) {
00059         fprintf(stderr,"%s:%d Unexpected and invalid parameters\n",__FILE__, __LINE__);
00060         abort();
00061     }
00062
00063     strftime(l_Time, sizeof(l_Time), "%Y-%m-%d %H:%M:%S", localtime(&l_CurrentTime));
00064
00065     /* Timestamp build with the filename, file line, function name */
00066     snprintf(l_tmp,sizeof(l_tmp),
00067             "%19s [%20s:%-4ud] (%11s @ %8s) %30s()",
00068             l_Time,p_File,p_Line,p_CompilDate,p_CompilTime,p_Function
00069             );
00070     return &l_tmp[0];
00071 }
00072
00073
00074 /* Documentation in header file */
00075 void _trace(const char* p_File,
00076            const unsigned int p_Line,
00077            const char* p_CompilDate,
00078            const char* p_CompilTime, const char* p_Function)
00079 {
00080     /* Parameter validity enforced by _timestamp */
00081     fprintf (stderr,"%s\n",
00082             _timestamp( p_File, p_Line, p_CompilDate, p_CompilTime, p_Function)
00083             );
00084 }
00085
00086 /* Documentation in header file */
00087 void _trace_msg(const char* p_File,
00088               const unsigned int p_Line,
00089               const char* p_CompilDate,
00090               const char* p_CompilTime,
00091               const char* p_Function, const char* p_Message)
00092 {
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
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 {
00108     /* Formatted message string */
00109     char l_tmp[100];
00110     /* Formatted message length */
00111     unsigned int l_length;
00112
00113     /* Limit variadic processing scope in a block */
00114     {
00115         /* Build the formatted message */
00116         va_list l_ap;
00117         va_start(l_ap, p_Format);
00118         l_length = vsnprintf(l_tmp, sizeof(l_tmp), p_Format, l_ap);
00119         va_end(l_ap);
00120     }
00121
00122     if (l_length >= sizeof(l_tmp)-1) {
00123         /* Indicate that message was truncated */
00124         strcpy(&l_tmp[sizeof(l_tmp)-6], "...");
00125     }
00126 }
00127
00128 /* Parameter validity enforced by _timestamp */
00129 fprintf (stderr,"%s : %s\n",
00130         _timestamp( p_File, p_Line, p_CompilDate, p_CompilTime, p_Function),
00131         l_tmp
00132         );
00133 }
00134

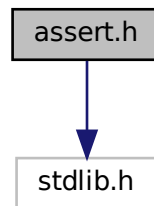
```

5.5 assert.h File Reference

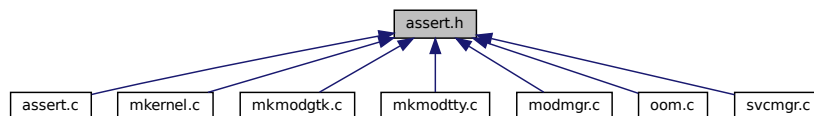
Debugging macros.

```
#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.

5.5.1 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](#).

5.5.2 Macro Definition Documentation

5.5.2.1 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](#).

5.5.2.2 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](#).

5.5.2.3 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](#).

5.5.2.4 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](#).

5.5.2.5 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](#).

5.5.3 Function Documentation

5.5.3.1 _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 75 of file [assert.c](#).

5.5.3.2 _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 sprintf function

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

5.5.3.3 _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 87 of file [assert.c](#).

5.6 assert.h

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

```

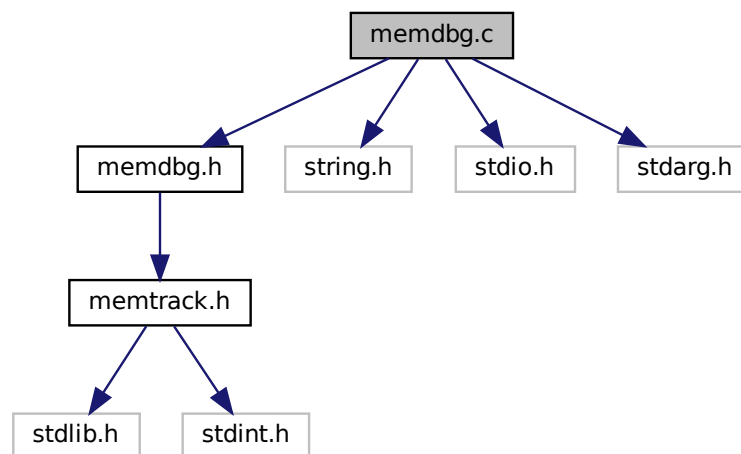
00001
00019 #ifndef __ASSERT_H__
00020 #define __ASSERT_H__
00021
00022 #ifdef HAVE_CONFIG_H
00023 #include "config.h"
00024 #endif
00025
00026 #include <stdlib.h> /* abort */
00027
00028 #ifndef NDEBUG
00029
00030 #ifdef __cplusplus
00031 extern "C" {
00032 #endif
00033
00045 void _trace(const char *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          /* NDEBUG */
00110 #define ASSERT(condition)
00111 #endif          /* NDEBUG */
00112
00118 #ifndef NDEBUG
00119 #define DBG_TRACE \
00120 _trace(__FILE__, __LINE__, __DATE__, __TIME__, __func__)
00121 #else          /* NDEBUG */
00122 #define DBG_TRACE
00123 #endif          /* NDEBUG */
00124
00130 #ifndef NDEBUG
00131 #define DBG_MSG(msg) \
00132 _trace_msg(__FILE__, __LINE__, __DATE__, __TIME__, __func__, msg)
00133 #else          /* NDEBUG */
00134 #define DBG_MSG(msg)
00135 #endif          /* NDEBUG */
00136
00142 #ifndef NDEBUG
00143 #define DBG_ITRACE(inst) \
00144 _trace_msg(__FILE__, __LINE__, __DATE__, __TIME__, __func__, #inst), \
00145 inst
00146 #else          /* NDEBUG */
00147 #define DBG_ITRACE(inst) inst
00148 #endif          /* NDEBUG */
00149
00155 #ifndef NDEBUG
00156 #define DBG_PRINTF(p_Format, ...) \
00157 _trace_dynmsg(__FILE__, __LINE__, __DATE__, __TIME__, __func__, p_Format, __VA_ARGS__)
00158 #else          /* NDEBUG */
00159 #define DBG_PRINTF(p_Format, ...)
00160 #endif          /* NDEBUG */
00161
00162 #endif          /* _ASSERT_H */
00163

```

5.7 memdbg.c File Reference

Memory leak tracker implementation.

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



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.

5.7.1 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](#).

5.7.2 Function Documentation

5.7.2.1 dbg_asprintf()

```
int dbg_asprintf (
    char ** p_Ptr,
    const char * p_Format,
    const char * File,
    const int Line,
    const char * CompileDate,
    const char * CompileTime,
    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>CompileDate</i>	: File compilation date
in	<i>CompileTime</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 196 of file [memdbg.c](#).

5.7.2.2 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 76 of file [memdbg.c](#).

Here is the call graph for this function:



5.7.2.3 dbg_free()

```

void dbg_free (
    void * Ptr,
    const char * File,
    const int Line,
    const char * CompileDate,
    const char * CompileTime,
    const char * Function )
  
```

Free compatible standard memory release.

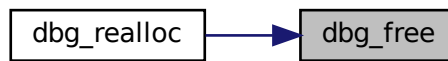
If the *Ptr* value is not NULL, try to untrack it. If it can not be found in the tracked list, because it was not tracked, it was allocated by a non monitored function, the function abort the process for investigation (missing a monitoring macro/function in the tracker, bug in the tracker, allocation 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>CompileDate</i>	: File compilation date
in	<i>CompileTime</i>	: File compilation time
in	<i>Function</i>	: Source function name

Definition at line 52 of file [memdbg.c](#).

Here is the caller graph for this function:



5.7.2.4 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
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

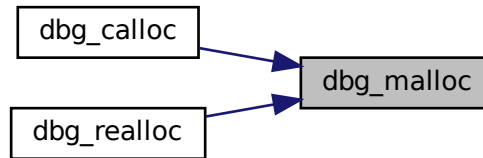
Allocated block address

Return values

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

Definition at line 25 of file [memdbg.c](#).

Here is the caller graph for this function:



5.7.2.5 `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

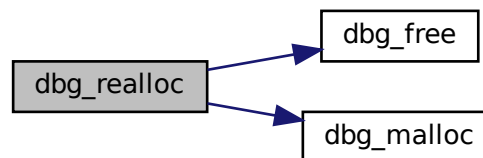
<i>Address</i>	if succeeded,
<i>NULL</i>	if not possible.

< Existing bloc size

< New block

Definition at line [102](#) of file [memdbg.c](#).

Here is the call graph for this function:

**5.7.2.6 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 163 of file `memdbg.c`.

5.8 memdbg.c

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

```

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

```

```

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

```

```

00163 char* dbg_strdup(
00164     const char* Ptr,
00165     const char* File,
00166     const int Line,
00167     const char* CompilDate,
00168     const char* CompilTime,
00169     const char* Function
00170 )
00171 {
00172     char* l_newblk;
00173     /* Use strdup to actually copy the string with its own return values and
00174     * abort (SIGSEGV in case of NULL) */
00175     l_newblk=NULL;
00176     l_newblk=strdup(Ptr);
00177
00178     /* If the copy succeeded, try to track the memory allocation */
00179     if (NULL != l_newblk)
00180         if (!memtrack_addblock(
00181             l_newblk,
00182             strlen(l_newblk)+1,
00183             File,Line,CompilDate,CompilTime,Function
00184         )) {
00185             /* If tracking fails, the whole operation is reverted and fails */
00186             free(l_newblk);
00187             l_newblk=NULL;
00188         };
00189     };
00190
00191     return l_newblk;
00192 }
00193
00194
00195 /* Documented in header file */
00196 int dbg_asprintf(char **p_Ptr,
00197     const char* p_Format,
00198     const char *File,
00199     const int Line,
00200     const char *CompilDate,
00201     const char *CompilTime, const char *Function,
00202     ...)
00203 {
00204     int l_returncode;
00205
00206     /* NULL is not allowed, where would we store the result, then ? */
00207     if(NULL==p_Ptr)
00208         abort();
00209
00210     /* Limit the scope of the variadic manipulation variables */
00211     {
00212         va_list l_ap;
00213         va_start (l_ap, Function);
00214         /* Use the original vasprintf to build the formatted string */
00215         l_returncode = vasprintf(p_Ptr, p_Format, l_ap);
00216         va_end(l_ap);
00217     }
00218
00219     /* If formatting succeeded, try to track the memory allocation */
00220     if (-1 != l_returncode)
00221         if (!memtrack_addblock(
00222             *p_Ptr,
00223             strlen (*p_Ptr)+1,
00224             File,Line,CompilDate,CompilTime,Function
00225         )) {
00226             /* If tracking fails, the whole operation is reverted and fails */
00227             free(*p_Ptr);
00228             l_returncode=-2;
00229         };
00230     };
00231
00232     return l_returncode;
00233 }
00234 }
00235

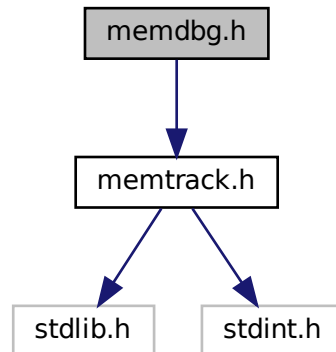
```

5.9 memdbg.h File Reference

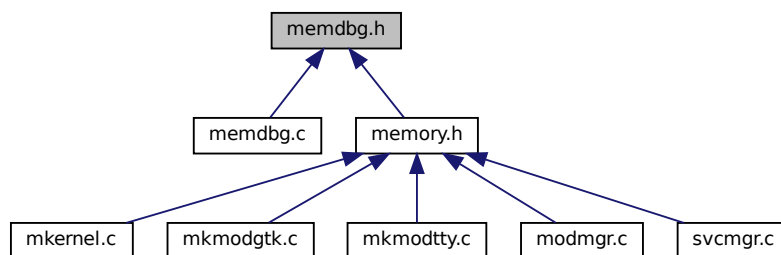
Memory leak tracker header.

```
#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 **CompileDate*, const char **CompileTime*, const char **Function*,...)

Build a formatted string with allocation.

5.9.1 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](#).

5.9.2 Function Documentation

5.9.2.1 `dbg_asprintf()`

```
int dbg_asprintf (
    char ** p_Ptr,
    const char * p_Format,
    const char * File,
    const int Line,
    const char * CompileDate,
    const char * CompileTime,
    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>CompileDate</i>	: File compilation date
in	<i>CompileTime</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 196 of file [memdbg.c](#).

5.9.2.2 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 76 of file [memdbg.c](#).

Here is the call graph for this function:



5.9.2.3 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 52 of file [memdbg.c](#).

Here is the caller graph for this function:



5.9.2.4 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
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

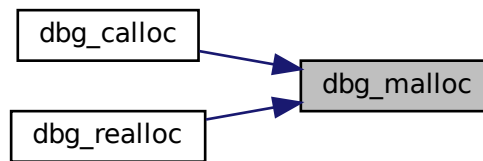
Allocated block address

Return values

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

Definition at line 25 of file [memdbg.c](#).

Here is the caller graph for this function:



5.9.2.5 `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

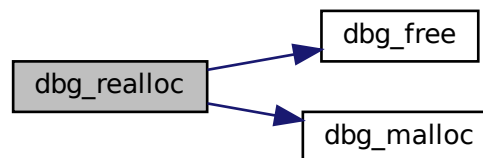
<i>Address</i>	if succeeded,
<i>NULL</i>	if not possible.

< Existing bloc size

< New block

Definition at line 102 of file [memdbg.c](#).

Here is the call graph for this function:

**5.9.2.6 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 163 of file `memdbg.c`.

5.10 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

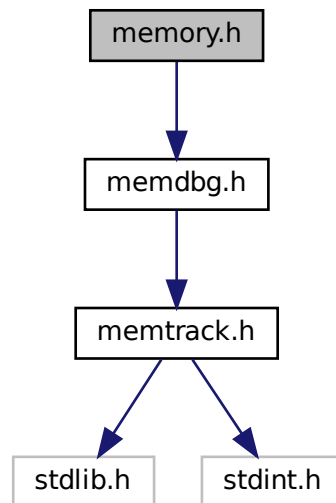
```

5.11 memory.h File Reference

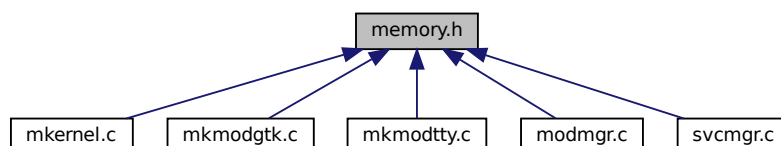
Tracks memory allocation and leaks when compiled without NDEBUG.

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

Include dependency graph for memory.h:



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



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.

5.11.1 Detailed Description

Tracks memory allocation and leaks when compiled without NDEBUG.

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](#).

5.11.2 Macro Definition Documentation

5.11.2.1 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](#).

5.11.2.2 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](#).

5.11.2.3 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](#).

5.11.2.4 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](#).

5.11.2.5 memreport

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

Prints a list of currently allocated blocks on stderr.

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

5.11.2.6 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](#).

5.11.2.7 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](#).

5.12 memory.h

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

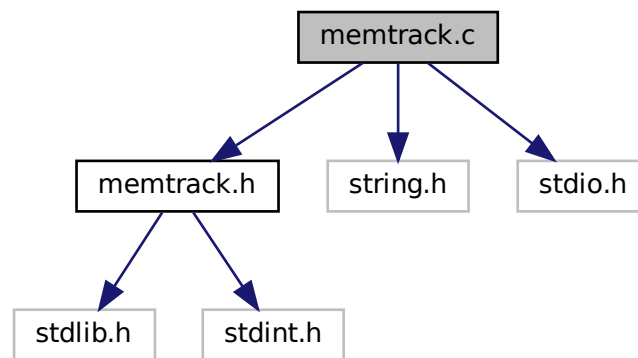
```
00001
00020 #ifndef __MEMORY_H__
00021 #define __MEMORY_H__
00022
00023 #ifdef HAVE_CONFIG_H
00024 # include "config.h"
00025 #endif
00026
00027 #ifndef NDEBUG
00028
00029 #include "memdbg.h"
00030
00032 #define malloc(size) dbg_malloc(size, __FILE__, __LINE__, __DATE__, __TIME__, __func__)
00033
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
```

5.13 memtrack.c File Reference

Memory block metadata tracking implementation.

```
#include "memtrack.h"
#include <string.h>
#include <stdio.h>
```

Include dependency graph for memtrack.c:



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 Function Documentation

5.13.2.1 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 [99](#) of file [memtrack.c](#).

5.13.3 Variable Documentation

5.13.3.1 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 )
```

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 [593](#) of file [memtrack.c](#).

5.13.3.2 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 )
```

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 603 of file [memtrack.c](#).

5.13.3.3 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 612 of file [memtrack.c](#).

5.13.3.4 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 615 of file [memtrack.c](#).

5.13.3.5 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 618 of file [memtrack.c](#).

5.13.3.6 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 [621](#) of file [memtrack.c](#).

5.14 memtrack.c

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

```
00001
00019 #include "memtrack.h"
00020 #include <string.h>          /* strdup */
00021 #include <stdio.h>          /* fprintf */
00022
00023 static struct MemBlock *Head;
00024 static struct MemBlock *Tail;
00025 static uint64_t NbBlocks;
00026 static uint64_t RAMSize;
00028 /* Early definition, documented in the implementation below */
00029 static unsigned int memtrack_init();
00030
00031 /* Early definition, documented in the implementation below */
00032 static unsigned int memtrack_addblock_preinit(const void *p_Ptr,
00033     const size_t p_Size,
00034     const char *p_File,
00035     const int p_Line,
00036     const char *p_CompilDate,
00037     const char *p_CompilTime,
00038     const char *p_Function);
00039
00040 /* Early definition, documented in the implementation below */
00041 static unsigned int memtrack_delblock_preinit(const void *p_Ptr,
00042     const char *p_File,
00043     const int p_Line,
00044     const char *p_CompilDate,
00045     const char *p_CompilTime,
00046     const char *p_Function);
00047
00048 /* Early definition, documented in the implementation below */
00049 static uint64_t memtrack_dumpblocks_preinit();
00050
00051 /* Early definition, documented in the implementation below */
00052 static uint64_t memtrack_getallocatedblocks_preinit();
00053
00054 /* Early definition, documented in the implementation below */
00055 static uint64_t memtrack_getallocatedRAM_preinit();
00056
```



```

00057 /* Early definition, documented in the implementation below */
00058 static size_t memtrack_getblocksize_preinit(const void *p_Ptr);
00059
00060 /* Early definition, documented in the implementation below */
00061 static unsigned int memtrack_addblock_postinit(const void *p_Ptr,
00062        const size_t p_Size,
00063        const char *p_File,
00064        const int p_Line,
00065        const char *p_CompilDate,
00066        const char *p_CompilTime,
00067        const char *p_Function);
00068
00069 /* Early definition, documented in the implementation below */
00070 static unsigned int memtrack_delblock_postinit(const void *p_Ptr,
00071        const char *p_File,
00072        const int p_Line,
00073        const char *p_CompilDate,
00074        const char *p_CompilTime,
00075        const char *p_Function);
00076
00077 /* Early definition, documented in the implementation below */
00078 static uint64_t memtrack_dumpblocks_postinit();
00079
00080 /* Early definition, documented in the implementation below */
00081 static uint64_t memtrack_getallocatedblocks_postinit();
00082
00083 /* Early definition, documented in the implementation below */
00084 static uint64_t memtrack_getallocatedRAM_postinit();
00085
00086 /* Early definition, documented in the implementation below */
00087 static size_t memtrack_getblocksize_postinit(const void *p_Ptr);
00088
00089 /*****
00090  * Implementations */
00091 /*****
00092
00099 void memtrack_reset()
00100 {
00101     /* Ne réinitialise que si nécessaire */
00102     if ( memtrack_addblock == memtrack_addblock_preinit)
00103         return;
00104
00105     /* Déconfiguration des foncteurs */
00106     memtrack_addblock = memtrack_addblock_preinit;
00107     memtrack_delblock = memtrack_delblock_preinit;
00108     memtrack_dumpblocks = memtrack_dumpblocks_preinit;
00109     memtrack_getallocatedblocks = memtrack_getallocatedblocks_preinit;
00110     memtrack_getallocatedRAM = memtrack_getallocatedRAM_preinit;
00111     memtrack_getblocksize = memtrack_getblocksize_preinit;
00112
00113     /* Purge de la liste */
00114     while (Tail!=Head->Next) {
00115         memtrack_delblock_postinit(Head->Next->Ptr, NULL, 0, NULL, NULL, NULL);
00116     }
00117
00118     /* Réinitialisation des pointeurs */
00119     free(Head);
00120     free(Tail);
00121     Head=NULL;
00122     Tail=NULL;
00123 }
00124
00141 static unsigned int memtrack_init()
00142 {
00143     Head = (TMemBlock *) malloc(sizeof(TMemBlock));
00144     Tail = (TMemBlock *) malloc(sizeof(TMemBlock));
00145
00146     if ((NULL==Head)|| (NULL==Tail)) {
00147         fprintf(stderr, "%s:%d Not enough memory to initialize memtracker\n",
00148             __FILE__, __LINE__);
00149         return 1;
00150     }
00151     Head->Prev = (TMemBlock *) NULL;
00152     Head->Next = Tail;
00153     Tail->Next = (TMemBlock *) NULL;
00154     Tail->Prev = Head;
00155     Tail->Ptr = Head->Ptr = (void *) NULL;
00156     Tail->Size = Head->Size = 0;
00157     Tail->File = Head->File = (char *) NULL;
00158     Tail->Line = Head->Line = 0;
00159     Tail->CompilDate = Head->CompilDate = (char *) NULL;
00160     Tail->CompilTime = Head->CompilTime = (char *) NULL;
00161     Tail->Function = Head->Function = (char *) NULL;
00162
00163     /* Initialisation des compteurs */
00164     NbBlocks=0;
00165     RAMSize=0;

```

```

00166
00167  /* Modification des foncteurs pour utiliser les fonctions définitives */
00168  memtrack_addblock = memtrack_addblock_postinit;
00169  memtrack_delblock = memtrack_delblock_postinit;
00170  memtrack_dumpblocks = memtrack_dumpblocks_postinit;
00171  memtrack_getallocatedblocks = memtrack_getallocatedblocks_postinit;
00172  memtrack_getallocatedRAM = memtrack_getallocatedRAM_postinit;
00173  memtrack_getblocksize = memtrack_getblocksize_postinit;
00174
00175  return 0;
00176 }
00177
00178
00190 static unsigned int memtrack_addblock_preinit(const void *p_Ptr,
00191      const size_t p_Size,
00192      const char *p_File,
00193      const int p_Line,
00194      const char *p_CompilDate,
00195      const char *p_CompilTime,
00196      const char *p_Function)
00197 {
00198  /* Initialisation de la liste */
00199  if (0!=memtrack_init()) {
00200      fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",
00201          __FILE__,__LINE__);
00202      /* OOM */
00203      return 1;
00204  }
00205
00206  /* Appel de la fonction réelle */
00207  return memtrack_addblock(p_Ptr, p_Size, p_File, p_Line, p_CompilDate,
00208      p_CompilTime, p_Function);
00209 }
00210
00222 static unsigned int memtrack_delblock_preinit(const void *p_Ptr,
00223      const char *p_File,
00224      const int p_Line,
00225      const char *p_CompilDate,
00226      const char *p_CompilTime,
00227      const char *p_Function)
00228 {
00229  /* Initialisation de la liste */
00230  if (0!=memtrack_init()) {
00231      fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",
00232          __FILE__,__LINE__);
00233      /* OOM */
00234      return 1;
00235  }
00236
00237  /* Appel de la fonction réelle */
00238  return memtrack_delblock(p_Ptr, p_File, p_Line, p_CompilDate,
00239      p_CompilTime, p_Function);
00240 }
00241
00253 static uint64_t memtrack_dumpblocks_preinit()
00254 {
00255  /* Initialisation de la liste */
00256  if (0!=memtrack_init()) {
00257      fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",
00258          __FILE__,__LINE__);
00259      /* OOM */
00260      return 0;
00261  }
00262
00263  /* Appel de la fonction réelle */
00264  return memtrack_dumpblocks();
00265 }
00266
00278 static uint64_t memtrack_getallocatedblocks_preinit()
00279 {
00280  /* Initialisation de la liste */
00281  if (0!=memtrack_init()) {
00282      fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",
00283          __FILE__,__LINE__);
00284      /* OOM */
00285      return 0;
00286  }
00287
00288  /* Appel de la fonction réelle */
00289  return memtrack_getallocatedblocks();
00290 }
00291
00303 static uint64_t memtrack_getallocatedRAM_preinit()
00304 {
00305  /* Initialisation de la liste */
00306  if (0!=memtrack_init()) {
00307      fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",

```

```

00308         __FILE__, __LINE__);
00309     /* OOM */
00310     return 0;
00311 }
00312
00313 /* Appel de la fonction réelle */
00314 return memtrack_getallocatedRAM();
00315 }
00316
00328 static size_t memtrack_getblocksize_preinit(const void *p_Ptr)
00329 {
00330     /* Initialisation de la liste */
00331     if (0!=memtrack_init()) {
00332         fprintf(stderr,"%s:%d Not enough memory to initialize memtracker\n",
00333             __FILE__, __LINE__);
00334         /* OOM */
00335         return 0;
00336     }
00337
00338     /* Appel de la fonction réelle */
00339     return memtrack_getblocksize(p_Ptr);
00340 }
00341
00350 static unsigned int memtrack_addblock_postinit(const void *p_Ptr,
00351     const size_t p_Size,
00352     const char *p_File,
00353     const int p_Line,
00354     const char *p_CompilDate,
00355     const char *p_CompilTime,
00356     const char *p_Function)
00357 {
00358     TMemBlock *l_tmp;
00359
00360     /* Test de validité des données à enregistrer */
00361     /* Meme si malloc permet Size=0, ce n'est pas portable */
00362     if ((NULL==p_Ptr)||
00363         (0==p_Size)||
00364         (NULL==p_File)||
00365         (0==p_File[0])||
00366         (0==p_Line)||
00367         (NULL==p_CompilDate)||
00368         (0==p_CompilDate[0])||
00369         (NULL==p_CompilTime)||
00370         (0==p_CompilTime[0])||
00371         (NULL==p_Function)||
00372         (0==p_Function[0])) {
00373         fprintf(stderr,"%s:%d Null or empty parameters\n",__FILE__, __LINE__);
00374         return 1;
00375     }
00376
00377     /* On ne peut pas dupliquer un pointeur. Pour le modifier, il faut le
00378 * supprimer et le recréer, ce n'est pas le rôle de ces fonctions de
00379 * bas-niveau */
00380
00381     /* Recherche du pointeur */
00382     l_tmp = Head->Next;
00383     while ((l_tmp->Ptr != p_Ptr) && (l_tmp != Tail))
00384         l_tmp = l_tmp->Next;
00385
00386     /* Le bloc ne doit pas avoir été trouvé */
00387     if (l_tmp != Tail) {
00388         fprintf(stderr,"%s:%d Memory bloc already registered\n",__FILE__, __LINE__);
00389         return 1;
00390     }
00391
00392     /* Allocation d'un nouveau descripteur de bloc */
00393     l_tmp = (TMemBlock *) malloc(sizeof(TMemBlock));
00394
00395     /* Allocation réussie ? */
00396     if (NULL == l_tmp) {
00397         return 1;
00398     }
00399
00400     /* Remplissage du descripteur */
00401     l_tmp->Ptr = (void *)p_Ptr;
00402     l_tmp->Size = p_Size;
00403     if (NULL==(l_tmp->File = strdup(p_File?p_File:""))) {
00404         free(l_tmp);
00405         return 1;
00406     };
00407     l_tmp->Line = p_Line;
00408     if (NULL==(l_tmp->CompilDate = strdup(p_CompilDate?p_CompilDate:""))) {
00409         free(l_tmp->File);
00410         free(l_tmp);
00411         return 1;
00412     };
00413     if (NULL==(l_tmp->CompilTime = strdup(p_CompilTime?p_CompilTime:""))) {

```

```

00414     free(l_tmp->CompilDate);
00415     free(l_tmp->File);
00416     free(l_tmp);
00417     return 1;
00418 };
00419 if (NULL==(l_tmp->Function = strdup(p_Function?p_Function:""))) {
00420     free(l_tmp->CompilTime);
00421     free(l_tmp->CompilDate);
00422     free(l_tmp->File);
00423     free(l_tmp);
00424     return 1;
00425 };
00426
00427 /* Ajout de la description dans la liste (Section critique) */
00428 l_tmp->Prev = Tail->Prev;
00429 l_tmp->Next = Tail;
00430 l_tmp->Prev->Next = l_tmp->Next->Prev = l_tmp;
00431
00432 /* Mise à jour des compteurs */
00433 NbBlocks++;
00434 RAMSize += p_Size;
00435
00436     return 0;
00437 }
00438
00447 static unsigned int memtrack_delblock_postinit(const void *p_Ptr,
00448     const char *p_File,
00449     const int p_Line,
00450     const char *p_CompilDate,
00451     const char *p_CompilTime,
00452     const char *p_Function)
00453 {
00454     TMemBlock *l_tmp;
00455     (void) p_File;
00456     (void) p_Line;
00457     (void) p_CompilDate;
00458     (void) p_CompilTime;
00459     (void) p_Function;
00460
00461     /* Recherche de la description */
00462     l_tmp = Head->Next;
00463     while ((l_tmp->Ptr != p_Ptr) && (l_tmp != Tail))
00464         l_tmp = l_tmp->Next;
00465
00466     /* Le bloc doit avoir été trouvé */
00467     if (l_tmp == Tail) {
00468         fprintf(stderr, "%s:%d Block not found for deletion\n", __FILE__, __LINE__);
00469         return 1;
00470     }
00471
00472     /* Libération des ressources acquises */
00473     /* On ne libère pas le bloc mémoire lui-même */
00474     free(l_tmp->File);
00475     free(l_tmp->CompilDate);
00476     free(l_tmp->CompilTime);
00477     free(l_tmp->Function);
00478
00479     /* Retrait de la description de la liste (Section critique) */
00480     l_tmp->Next->Prev = l_tmp->Prev;
00481     l_tmp->Prev->Next = l_tmp->Next;
00482
00483     /* Mise à jour des compteurs */
00484     NbBlocks--;
00485     RAMSize -= l_tmp->Size;
00486
00487     /* Libération de la description */
00488     free(l_tmp);
00489
00490     return 0;
00491 }
00492
00501 static uint64_t memtrack_dumpblocks_postinit()
00502 {
00503     TMemBlock *l_tmp;
00504     uint64_t l_NbBlocks = 0;;
00505
00506     if (Head->Next != Tail) {
00507         size_t l_BlockSize;
00508         fprintf(stderr,
00509
00510             "+-----+\n");
00510             fprintf(stderr, "| %-10s |\n", "Memory Tracker Report");
00511             fprintf(stderr,
00512
00513             "+-----+\n");
00513             fprintf(stderr,
00514                 "| %-20s | %-20s | %-4s | %-15s | %-8s | %-22s |\n",

```

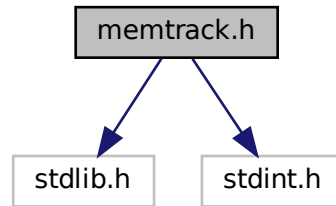

5.15 memtrack.h File Reference

Memory block metadata tracking headers.

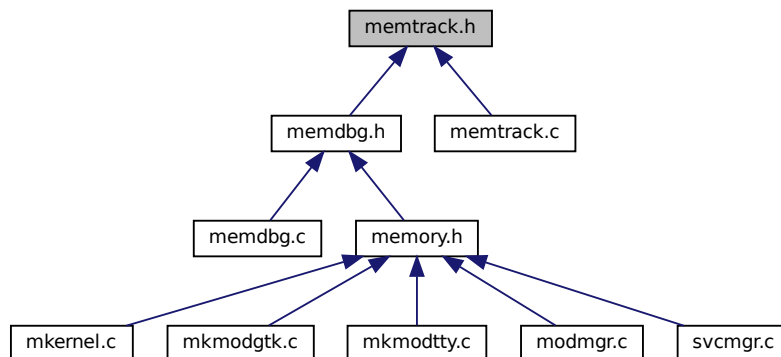
```
#include <stdlib.h>
```

```
#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.

5.15.1 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](#).

5.15.2 Typedef Documentation

5.15.2.1 TMemBlock

```
typedef struct MemBlock TMemBlock
```

Memory block metadata list item.

Double linked list item to store memory block metadata

5.15.3 Variable Documentation

5.15.3.1 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 593 of file [memtrack.c](#).

5.15.3.2 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 603 of file [memtrack.c](#).

5.15.3.3 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 612 of file [memtrack.c](#).

5.15.3.4 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 615 of file [memtrack.c](#).

5.15.3.5 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 618 of file [memtrack.c](#).

5.15.3.6 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 621 of file [memtrack.c](#).

5.16 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
00029 #ifdef __cplusplus
00030 extern "C" {
00031 #endif
00032
00037 typedef struct MemBlock {
00038     struct MemBlock *Prev;
00039     struct MemBlock *Next;
00040     void *Ptr;
00041     size_t Size;
00042     char *File;
00043     int Line;
00044     char *CompileDate;
00045     char *CompileTime;
00046     char *Function;
00047 } TMemBlock;
00048
00079 extern unsigned int (*memtrack_addblock) (const void *p_Ptr,
00080     const size_t p_Size,
00081     const char *p_File,
00082     const int p_Line,
00083     const char *p_CompileDate,
00084     const char *p_CompileTime,
00085     const char *p_Function);
00086
00115 extern unsigned int (*memtrack_delblock) (const void *p_Ptr,
00116     const char *p_File,
00117     const int p_Line,
```

```

00118     const char *p_CompilDate,
00119     const char *p_CompilTime,
00120     const char *p_Function);
00121
00135 extern uint64_t (*memtrack_dumpblocks) ();
00136
00149 extern uint64_t (*memtrack_getallocatedblocks) ();
00150
00163 extern uint64_t (*memtrack_getallocatedRAM) ();
00164
00182 extern size_t (*memtrack_getblocksize) (const void *p_Ptr);
00183
00184 #ifdef __cplusplus
00185 }
00186 #endif
00187
00188 #endif                                     /* __MEMTRACK_H__ */
00189

```

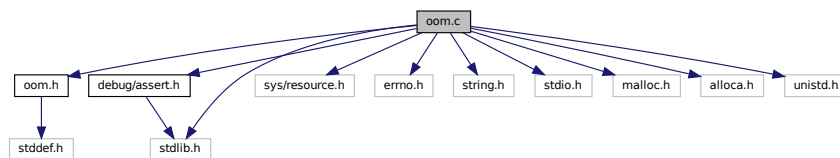
5.17 oom.c File Reference

```

#include "oom.h"
#include "debug/assert.h"
#include <sys/resource.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <alloca.h>
#include <unistd.h>

```

Include dependency graph for oom.c:



Macros

- `#define __asm__ asm`
- `#define __sync_synchronize void`
- `#define _GNU_SOURCE`
- `#define RAMBLOCKS_MAX 1000`

Maximum number of fragmented blocks to allocate.

Functions

- `size_t oomtest_config (const size_t hardlimit)`
Sets the oomtest helpers hard rlimit and enables the oomtest helper features.
- `unsigned char oomtest_enabled ()`

Variables

- `size_t(* oomtest_fill)(const size_t minHeap, const size_t minStack) =oomtest_fill_preinit`
Starts an almost OOM single and simple test.
- `void(* oomtest_free)() =oomtest_free_preinit`
Ends a single simple OOM test.
- `size_t(* oomtest_enable)(const size_t softlimit) =oomtest_enable_preinit`
Starts a new oomtest environment or reconfigure the soft limit.
- `size_t(* oomtest_disable)() =oomtest_disable_preinit`
Stops the current oomtest.

5.17.1 Detailed Description

Author

François Cerbelle (Fanfan), francois@cerbelle.net

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

5.17.2 Macro Definition Documentation

5.17.2.1 `__asm__`

```
#define __asm__ asm
```

Definition at line 21 of file [oom.c](#).

5.17.2.2 `__sync_synchronize`

```
#define __sync_synchronize void
```

Definition at line 24 of file [oom.c](#).

5.17.2.3 `_GNU_SOURCE`

```
#define _GNU_SOURCE
```

Definition at line 38 of file [oom.c](#).

5.17.2.4 RAMBLOCKS_MAX

```
#define RAMBLOCKS_MAX 1000
```

Maximum number of fragmented blocks to allocate.

I never reached more than 350 on my systems. The value needs to be less than `UINT_MAX`

Definition at line 47 of file [oom.c](#).

5.17.3 Function Documentation

5.17.3.1 oomtest_config()

```
size_t oomtest_config (  
    const size_t hardlimit )
```

Sets the oomtest helpers hard rlimit and enables the oomtest helper features.

This function needs to be invoked BEFORE any other helper from the framework. It will configure an hard RAM limit for the current process and each of his children. Then, it will enable the other oomtest helper functions.

It refuses to set a limit over the actually installed physical RAM to limit pushing RAM pages to the swap. If the requested value is zero, it will default to the actually installed physical RAM.

If anything goes wrong or did not behaves as expected, it abort the process to avoid RAM bombing and swapping.

Despite it was designed to be invoked only once, from the main parent process, it can be invoked several times as long as the *hardlimit* parameter is always less than the previous call, otherwise it will fail and abort the process.

Parameters

<code>in</code>	<i>hardlimit</i>	the size in bytes to limit the process to.
-----------------	------------------	--

Returns

Returns the actual configured size. It should be the same value as the *hardlimit* parameter, otherwise something went wrong, it was not detected and the process was not aborted (which is a bug to report)

See also

[oomtest_enable](#)

[oomtest_disable](#)

[oomtest_fill](#)

[oomtest_free](#)

Definition at line 367 of file [oom.c](#).

5.17.3.2 oomtest_enabled()

```
unsigned char oomtest_enabled ( )
```

Definition at line 422 of file [oom.c](#).

5.17.4 Variable Documentation

5.17.4.1 oomtest_disable

```
size_t(* oomtest_disable) ( ) ( ) =oomtest_disable_preinit
```

Stops the current oomtest.

This function can be invoked any time after the oomtest_config initialized the environment. If invoked before, it will abort the current process to help detecting mistakes in the test code.

This function disables the fill/free oomtest functions and free the allocated RAM before reverting the soft limit to the hard limit value.

It is designed to be invoked at the end of a test case.

Despite it could be invoked twice or without a prior call to oomtest_enable, it is illegal and not allowed to help detecting mistakes in the test code.

Returns

Returns the applied soft limit, which should be the same as the hard limit.

See also

[oomtest_enable](#)

[oomtest_disable](#)

[oomtest_fill](#)

[oomtest_free](#)

Definition at line 138 of file [oom.c](#).

5.17.4.2 oomtest_enable

```
size_t(* oomtest_enable) (const size_t softlimit) (  
    const size_t softlimit ) =oomtest_enable_preinit
```

Starts a new oomtest environment or reconfigure the soft limit.

This function can only be invoked after at least a first call to oomtest_config to initialize the oomtest helpers environment. If invoked before, it will abort the current process to help detecting mistakes in the test code.

It will (soft)limit the current process and his children to the provided value in bytes. Then, it will enable the fill/free oomtest helper functions which are disabled otherwise.

It is designed to be invoked at the beginning of a test case to apply for all tests in this testcase.

In case of any unexpected behavior, it should abort the current process.

Parameters

in	<i>softlimit</i>	Soft limit to set in bytes. It has to be less than the hardlimit otherwise it will fail and abort. If set to 0, it will apply the same value as the hardlimit.
----	------------------	--

Returns

Returns the actually set value as a softlimit, either the requested value or the hardlimit in case of 0 requested.

See also

[oomtest_enable](#)

[oomtest_disable](#)

[oomtest_fill](#)

[oomtest_free](#)

Definition at line 120 of file [oom.c](#).

5.17.4.3 oomtest_fill

```
size_t(* oomtest_fill) (const size_t minHeap, const size_t minStack) (
    const size_t minHeap,
    const size_t minStack ) =oomtest_fill_preinit
```

Starts an almost OOM single and simple test.

This function is only enabled after a call to [oomtest_enable](#) and will have no effect otherwise. Its goal is to completely fill the RAM until the very last bytes, to keep only between minHeap and maxHeap bytes available in the heap and minStack bytes in the stack.

If [oomtest_enable](#) was not invoked beforehand, this function will simply return without any RAM consumption.

It should be called immediately before the test and should be reverted with [oomtest_free](#) immediately after. It can create so much pressure on the available memory that a simple printf could fail.

It is designed to fail if invoked twice as this is very probably a mistake in the test code. Should you need to change the RAM filling values, first call [oomtest_free](#) and reapply [oomtest_fill](#). This ensure a really wanted behavior and not a mistake in your test code.

Parameters

in	<i>minHeap</i>	
in	<i>minStack</i>	

Returns

Returns the allocated size to fill the memory

See also

[oomtest_enable](#)
[oomtest_disable](#)
[oomtest_fill](#)
[oomtest_free](#)

Definition at line 84 of file [oom.c](#).

5.17.4.4 oomtest_free

```
void(* oomtest_free) () ( ) =oomtest_free_preinit
```

Ends a single simple OOM test.

If this function is invoked between an `oomtest_enable` and an `oomtest_disable` invocations, it deallocates the RAM allocated by the `oomtest_fill` function. It should be called immediately after the single and simple test because the RAM pressure can even make a `printf` to fail.

If called without a prior `oomtest_enable` invocation, this function simply returns without any action.

This function is designed to fail and abort the process if invoked whereas there is no current RAM allocated, when called twice, for example. This helps to avoid mistakes in the test scenario.

Definition at line 100 of file [oom.c](#).

5.18 oom.c

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

```

00001
00017 #include "oom.h"                                /* OOM simulation */
00018
00019 #ifndef __GNUC__
00020 #   ifndef __asm__
00021 #       define __asm__ asm
00022 #   endif
00023 #   ifndef __sync_synchronize
00024 #       define __sync_synchronize void
00025 #   endif
00026 #endif
00027
00028 #include "debug/assert.h"                          /* libdebug's Assertions */
00029 #include <sys/resource.h>                          /* setrlimit */
00030 #include <errno.h>                                /* errno */
00031 #include <string.h>                               /* strerror() */
00032 #include <stdio.h>                               /* printf */
00033 #include <malloc.h>                              /* malloc_trim() */
00034 #include <stdlib.h>                              /* abort() */
00035 #include <alloca.h>                              /* alloca() */
00036
00037 #ifndef _GNU_SOURCE
00038 #define _GNU_SOURCE
00039 #endif
00040 #include <unistd.h>                              /* sysconf() */
00041
00047 #define RAMBLOCKS_MAX 1000
00048
00052 static void* _oomblocks[RAMBLOCKS_MAX] = {0};
00053
00055 static int checked_getrlimit(int resource, struct rlimit *rlim)
00056 {
00057     /* Get current limit values */
00058     if (getrlimit(resource, rlim) != 0) {

```

```

00059     /* Can occur, thus not ignored, but impossible to trigger for gcov/lcov */
00060     fprintf(stderr, "%s:%d getrlimit() failed with errno=%d %s\n",
00061             __FILE__, __LINE__, errno, strerror(errno));
00062     abort();
00063 }
00064 return 0;
00065 }
00066
00075 static size_t oomtest_fill_preinit(const size_t minHeap, const size_t minStack)
00076 {
00077     (void)minHeap;
00078     (void)minStack;
00079     ASSERT(NULL==_oomblocks[0]);
00080     return 0;
00081 }
00082
00083 /* Documented in header file */
00084 size_t (*oomtest_fill)(const size_t minHeap, const size_t minStack)=oomtest_fill_preinit;
00085
00094 static void oomtest_free_preinit()
00095 {
00096     ASSERT(NULL==_oomblocks[0]);
00097 }
00098
00099 /* Documented in header file */
00100 void (*oomtest_free)()=oomtest_free_preinit;
00101
00110 static size_t oomtest_enable_preinit(const size_t softlimit)
00111 {
00112     (void)softlimit;
00113
00114     /* Should not be called without configuration */
00115     fprintf(stderr, "%s:%d oomtest_enable called without oomtest_config before\n", __FILE__, __LINE__);
00116     abort();
00117 }
00118
00119 /* Documented in header file */
00120 size_t (*oomtest_enable)(const size_t softlimit)=oomtest_enable_preinit;
00121
00130 static size_t oomtest_disable_preinit()
00131 {
00132     /* Should not be called before configuration */
00133     fprintf(stderr, "%s:%d oomtest_disable called without oomtest_config before\n", __FILE__, __LINE__);
00134     abort();
00135 }
00136
00137 /* Documented in header file */
00138 size_t (*oomtest_disable)()=oomtest_disable_preinit;
00139
00151 static size_t oomtest_getbiggestblock(void** p_ramblock)
00152 {
00153     /* This function could receive an optimized "max" value and return the final
00154     * "max" value to the caller. So, it could be used as the next invocation
00155     * starting "max" value. This would save few loop iterations, at the cost of
00156     * extra stack usage, I chose to not pass this value as a parameter to avoid
00157     * consuming stack.
00158     * The function starts to search between 0..rlim_cur which has very little
00159     * impact given the Olog2 complexity. It could be optimized and start to
00160     * search between 0..sysconf(AVPHYSPAGE*PAGESIZE), assuming that sysconf is
00161     * faster than few loop iterations. */
00162
00163     /* Use static to avoid stack allocation/free */
00164     static size_t max, cur;
00165     static struct rlimit limit;
00166
00167     ASSERT(NULL!=p_ramblock);
00168     ASSERT(NULL==*p_ramblock);
00169
00170     /* Get the current limits */
00171     checked_getrlimit(RLIMIT_AS, &limit);
00172     max = limit.rlim_cur;
00173
00174     /* Restart the whole process if cur can not be allocated at the end */
00175     while ((max>0)&&(NULL==*p_ramblock)) {
00176         static size_t min;
00177         /* Iterate quickly (Olog2) to converge to the biggest available RAM block */
00178         min = 0;
00179         while (max>min) {
00180             cur = min+(max-min)/2; /* To avoid overflow */
00181             if (NULL==(p_ramblock = malloc(cur))) {
00182                 max = cur;
00183             } else {
00184                 min = cur+1;
00185                 free(*p_ramblock);
00186             }
00187         }
00188         cur -= 1;

```

```

00189     *p_ramblock=malloc(cur);
00190     }
00191
00192     return cur;
00193 }
00194
00202 static size_t oomtest_fill_postinit(const size_t minHeap, const size_t minStack)
00203 {
00204     unsigned int l_numblock;
00205     size_t l_sum;
00206     void* volatile l_reservedheap;
00207     void* l_reservedstack;
00208
00209     /* Probably a mistake in the test code, do not accept despite we could */
00210     if (NULL!=_oomblocks[0]) {
00211         /* Dirty hack to free some RAM and allow abort to SIGABRT */
00212         free(_oomblocks[0]);
00213         fprintf(stderr,"%s:%d oomblocks are already allocated\n", __FILE__, __LINE__);
00214         abort();
00215     }
00216
00217     /* Reserve/Protect stack bytes which will be auto freed at return */
00218     /* A failure means a stackoverflow, which is not recoverable and will abort
00219  * the process anyway. */
00220     if (0<minStack)
00221         l_reservedstack=alloca(minStack);
00222     (void)l_reservedstack;
00223
00224     /* Reserve/Protect heap bytes which will be released before return */
00225     l_reservedheap=NULL;
00226     if (0<minHeap)
00227         if (NULL==(l_reservedheap=malloc(minHeap))) {
00228             /* This will fail if minHeap is higher than rlimit */
00229             fprintf(stderr,"%s:%d Failed to reserve minheap bytes\n",__FILE__, __LINE__);
00230             abort();
00231         }
00232
00233     /* Find and allocate the biggest available RAM blocks until no mre RAM
00234  * available or all _oomblocks are allocated */
00235     l_numblock=0;
00236     l_sum = 0;
00237     l_sum += oomtest_getbiggestblock(&(_oomblocks[l_numblock++]));
00238     while ((NULL!=_oomblocks[l_numblock-1])&&((RAMBLOCKS_MAX-1)>l_numblock))
00239         l_sum += oomtest_getbiggestblock(&(_oomblocks[l_numblock++]));
00240     /* Either already NULL, which stopped the while loop or reached the last
00241  * slot in the table, which stopped the while loop and the slot has to be
00242  * set to NULL */
00243     _oomblocks[l_numblock]=NULL;
00244
00245     /* There can be less than 4 bytes available at this point !!! */
00246
00247     /* Make the protected heap bytes available again */
00248     if (NULL!=l_reservedheap)
00249         free(l_reservedheap);
00250
00251     return l_sum;
00252 }
00253
00261 static void oomtest_free_postinit()
00262 {
00263     unsigned int l_i;
00264
00265     /* Despite we could manage, abort to help detecting mistakes in test code */
00266     if (NULL==_oomblocks[0]) {
00267         fprintf(stderr,"%s:%d no blocks to free in oomtest_free.\n",__FILE__, __LINE__);
00268         abort();
00269     }
00270
00271     /* Actually free allocated blocks and set their pointer to NULL */
00272     l_i = 0;
00273     while ((l_i<(RAMBLOCKS_MAX-1))&&(_oomblocks[l_i])) {
00274         free(_oomblocks[l_i]);
00275         _oomblocks[l_i] = NULL;
00276         l_i+=1;
00277     }
00278     /* These barrier protections are probably useless */
00279     malloc_trim(0);
00280     /* SW barrier (compiler only) */
00281     __asm__ volatile("": : : "memory");
00282     /* HW barrier (CPU instruction) */
00283     __sync_synchronize();
00284 }
00285
00293 static size_t oomtest_enable_postinit(const size_t softlimit)
00294 {
00295     struct rlimit limit;
00296     size_t l_softlimit = softlimit;

```

```

00297
00298 /* Probably a bug in oomtest */
00299 ASSERT(NULL==_oomblocks[0]);
00300
00301 /* Get current limit values */
00302 checked_getrlimit(RLIMIT_AS, &limit);
00303
00304 /* Check the requested value */
00305 if (0==l_softlimit) {
00306     /* Defaults to available physical RAM if requested 0 */
00307     l_softlimit = limit.rlim_max;
00308 };
00309
00310 /* Soft limit available RAM */
00311 limit.rlim_cur = l_softlimit;
00312
00313 /* Abort if setrlimit fails to avoid RAM bombing */
00314 if (setrlimit(RLIMIT_AS, &limit) != 0) {
00315     fprintf(stderr, "%s:%d setrlimit(cur=%lu, max=%lu) with errno=%d %s\n",
00316             __FILE__, __LINE__,
00317             (unsigned long)limit.rlim_cur, (unsigned long)limit.rlim_max,
00318             errno, strerror(errno));
00319     abort();
00320 }
00321
00322 /* Activate fill/free functions */
00323 oomtest_fill = oomtest_fill_postinit;
00324 oomtest_free = oomtest_free_postinit;
00325
00326 /* Return the actual current soft limit */
00327 return limit.rlim_cur;
00328 }
00329
00337 static size_t oomtest_disable_postinit()
00338 {
00339     struct rlimit limit;
00340
00341     if (NULL!=_oomblocks[0]) {
00342         fprintf(stderr, "%s:%d RAM still allocated while calling oomtest_disable\n", __FILE__, __LINE__);
00343         abort();
00344     }
00345
00346     /* Do not allow calling disable if not enabled to detect test mistakes */
00347     if ((oomtest_fill!=oomtest_fill_postinit)|| (oomtest_free!=oomtest_free_postinit)) {
00348         fprintf(stderr, "%s:%d Impossible to disable oomtest if not previously
00349         enabled\n", __FILE__, __LINE__);
00350         abort();
00351     }
00352
00353     /* Reset the soft limit to hard limit */
00354     oomtest_enable_postinit(0);
00355
00356     /* Get current limit values */
00357     checked_getrlimit(RLIMIT_AS, &limit);
00358
00359     /* Restore disabled functors */
00360     oomtest_fill = oomtest_fill_preinit;
00361     oomtest_free = oomtest_free_preinit;
00362
00363     /* Return the actual current soft limit, which is equal to hard limit */
00364     return limit.rlim_cur;
00365 }
00366
00367 /* Documented in header file */
00368 size_t oomtest_config(const size_t hardlimit)
00369 {
00370     struct rlimit limit;
00371     size_t l_avail;
00372     size_t l_hardlimit = hardlimit;
00373
00374     /* Probably a test implementation mistake */
00375     if (NULL!=_oomblocks[0]) {
00376         fprintf(stderr, "%s:%d Calling oomtest_config with allocated RAM blocks is not allowed.\n",
00377             __FILE__, __LINE__);
00378         abort();
00379     }
00380
00381     /* Find *installed* physical RAM, 0 in case of failure */
00382     l_avail = (sysconf(_SC_PHYS_PAGES) * sysconf(_SC_PAGESIZE));
00383
00384     /* Get current limit values */
00385     checked_getrlimit(RLIMIT_AS, &limit);
00386     if (0==l_hardlimit) {
00387         /* Defaults to available physical RAM or already set rlimit
00388         * if 0 requested */
00389         l_hardlimit=(limit.rlim_max<l_avail?limit.rlim_max:l_avail);
00390     } else if (l_hardlimit>l_avail) {

```

```

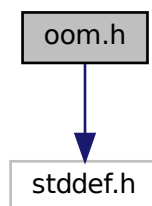
00390     /* Fails if request is over available physical RAM to avoid swapping */
00391     fprintf(stderr,"%s:%d Requesing a limit %lu bigger than *installed* RAM %lu is not
allowed.\n",
00392         __FILE__, __LINE__,
00393         (unsigned long)l_hardlimit, (unsigned long)l_avail);
00394     abort();
00395 }
00396
00397 /* Hard limit available RAM to hardlimit globally with no way back */
00398 limit.rlim_cur = l_hardlimit;
00399 limit.rlim_max = l_hardlimit;
00400
00401 /* Abort if setrlimit fails to avoid RAM bombing */
00402 if (setrlimit(RLIMIT_AS, &limit) != 0) {
00403     fprintf (stderr,"%s:%d setrlimit(cur=%lu, max=%lu) with errno=%d %s\n",
00404         __FILE__, __LINE__, (unsigned long)limit.rlim_cur,
00405         (unsigned long)limit.rlim_max, errno, strerror(errno));
00406     /* Get current limit values */
00407     checked_getrlimit(RLIMIT_AS, &limit);
00408     fprintf (stderr,"%s:%d getrlimit() is cur=%lu, max=%lu\n",
00409         __FILE__, __LINE__, (unsigned long)limit.rlim_cur,
00410         (unsigned long)limit.rlim_max);
00411     abort();
00412 }
00413
00414 /* Activate enable/disable functions */
00415 oomtest_enable = oomtest_enable_postinit;
00416 oomtest_disable = oomtest_disable_postinit;
00417
00418 /* Return the configured limit hard=soft */
00419 return limit.rlim_max;
00420 }
00421
00422 unsigned char oomtest_enabled()
00423 {
00424     return (oomtest_fill==oomtest_fill_postinit?1:0);
00425 }

```

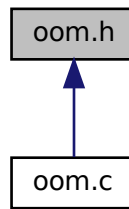
5.19 oom.h File Reference

#include <stddef.h>

Include dependency graph for oom.h:



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



Macros

- `#define RAMLIMIT_HARD (7*1024*1024)`
- `#define RAMLIMIT_SOFT (5*1024*1024)`

Functions

- `size_t oomtest_config (const size_t hardlimit)`
Sets the oomtest helpers hard rlimit and enables the oomtest helper features.
- `unsigned char oomtest_enabled ()`

Variables

- `size_t(* oomtest_enable)(const size_t softlimit)`
Starts a new oomtest environment or reconfigure the soft limit.
- `size_t(* oomtest_disable)()`
Stops the current oomtest.
- `size_t(* oomtest_fill)(const size_t minHeap, const size_t minStack)`
Starts an almost OOM single and simple test.
- `void(* oomtest_free)()`
Ends a single simple OOM test.

5.19.1 Detailed Description

Author

François Cerbelle (Fanfan), francois@cerbelle.net

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

5.19.2 Macro Definition Documentation

5.19.2.1 RAMLIMIT_HARD

```
#define RAMLIMIT_HARD (7*1024*1024)
```

Definition at line 25 of file [oom.h](#).

5.19.2.2 RAMLIMIT_SOFT

```
#define RAMLIMIT_SOFT (5*1024*1024)
```

Definition at line 28 of file [oom.h](#).

5.19.3 Function Documentation

5.19.3.1 oomtest_config()

```
size_t oomtest_config (  
    const size_t hardlimit )
```

Sets the oomtest helpers hard rlimit and enables the oomtest helper features.

This function needs to be invoked BEFORE any other helper from the framework. It will configure an hard RAM limit for the current process and each of his children. Then, it will enable the other oomtest helper functions.

It refuses to set a limit over the actually installed physical RAM to limit pushing RAM pages to the swap. If the requested value is zero, it will default to the actually installed physical RAM.

If anything goes wrong or did not behaves as expected, it abort the process to avoid RAM bombing and swapping.

Despite it was designed to be invoked only once, from the main parent process, it can be invoked several times as long as the *hardlimit* parameter is always less than the previous call, otherwise it will fail and abort the process.

Parameters

in	<i>hardlimit</i>	the size in bytes to limit the process to.
----	------------------	--

Returns

Returns the actual configured size. It should be the same value as the *hardlimit* parameter, otherwise something went wrong, it was not detected and the process was not aborted (which is a bug to report)

See also

[oomtest_enable](#)
[oomtest_disable](#)
[oomtest_fill](#)
[oomtest_free](#)

Definition at line [367](#) of file [oom.c](#).

5.19.3.2 oomtest_enabled()

```
unsigned char oomtest_enabled ( )
```

Definition at line [422](#) of file [oom.c](#).

5.19.4 Variable Documentation**5.19.4.1 oomtest_disable**

```
size_t(* oomtest_disable) ( ) ( ) [extern]
```

Stops the current oomtest.

This function can be invoked any time after the `oomtest_config` initialized the environment. If invoked before, it will abort the current process to help detecting mistakes in the test code.

This function disables the fill/free oomtest functions and free the allocated RAM before reverting the soft limit to the hard limit value.

It is designed to be invoked t the end of a test case.

Despite it could be invoked twice or without a prior call to `oomtest_enable`, it is illegal and not allowed to help detecting mistakes in the test code.

Returns

Returns the applied soft limit, which should be the same as the hard limit.

See also

[oomtest_enable](#)
[oomtest_disable](#)
[oomtest_fill](#)
[oomtest_free](#)

Definition at line [138](#) of file [oom.c](#).

5.19.4.2 oomtest_enable

```
size_t(* oomtest_enable) (const size_t softlimit) (
    const size_t softlimit ) [extern]
```

Starts a new oomtest environment or reconfigure the soft limit.

This function can only be invoked after at least a first call to oomtest_config to initialize the oomtest helpers environment. If invoked before, it will abort the current process to help detecting mistakes in the test code.

It will (soft)limit the current process and his children to the provided value in bytes. Then, it will enable the fill/free oomtest helper functions which are disabled otherwise.

It is designed to be invoked at the beginning of a test case to apply for all tests in this testcase.

In case of any unexpected behavior, it should abort the current process.

Parameters

in	<i>softlimit</i>	Soft limit to set in bytes. It has to be less than the hardlimit otherwise it will fail and abort. If set to 0, it will apply the same value as the hardlimit.
----	------------------	--

Returns

Returns the actually set value as a softlimit, either the requested value or the hardlimit in case of 0 requested.

See also

[oomtest_enable](#)
[oomtest_disable](#)
[oomtest_fill](#)
[oomtest_free](#)

Definition at line 120 of file [oom.c](#).

5.19.4.3 oomtest_fill

```
size_t(* oomtest_fill) (const size_t minHeap, const size_t minStack) (
    const size_t minHeap,
    const size_t minStack ) [extern]
```

Starts an almost OOM single and simple test.

This function is only enabled after a call to oomtest_enable and will have no effect otherwise. Its goal is to completely fill the RAM until the very last bytes, to keep only between minHeap and maxHeap bytes available in the heap and minStack bytes in the stack.

If oomtest_enable was not invoked beforehand, this function will simply return without any RAM consumption.

It should be called immediately before the test and should be reverted with oomtest_free immediately after. It can create so much pressure on the available memory that a simple printf could fail.

It is designed to fail if invoked twice as this is very probably a mistake in the test code. Should you need to change the RAM filling values, first call oomtest_free and reapply oomtest_fill. This ensure a really wanted behavior and not a mistake in your test code.

Parameters

in	<i>minHeap</i>	
in	<i>minStack</i>	

Returns

Returns the allocated size to fill the memory

See also

[oomtest_enable](#)

[oomtest_disable](#)

[oomtest_fill](#)

[oomtest_free](#)

Definition at line 84 of file [oom.c](#).

5.19.4.4 oomtest_free

```
void(* oomtest_free) () ( ) [extern]
```

Ends a single simple OOM test.

If this function is invoked between an `oomtest_enable` and an `oomtest_disable` invocations, it deallocates the RAM allocated by the `oomtest_fill` function. It should be called immediately after the single and simple test because the RAM pressure can even make a `printf` to fail.

If called without a prior `oomtest_enable` invocation, this function simply returns without any action.

This function is designed to fail and abort the process if invoked whereas there is no current RAM allocated, when called twice, for example. This helps to avoid mistakes in the test scenario.

Definition at line 100 of file [oom.c](#).

5.20 oom.h

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

```
00001
00017 #ifndef __OOM_H__
00018 #define __OOM_H__
00019
00020 #ifdef HAVE_CONFIG_H
00021 #include "config.h"
00022 #endif
00023
00024 #ifndef RAMLIMIT_HARD
00025 #define RAMLIMIT_HARD (7*1024*1024)
00026 #endif
00027 #ifndef RAMLIMIT_SOFT
00028 #define RAMLIMIT_SOFT (5*1024*1024)
00029 #endif
00030
00031 #include <stddef.h>                                /* size_t */
```

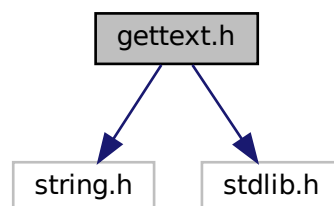
```
00032
00033 /*      void myfunction() {
00034 *          void* ptr;
00035 *          ptr = NULL;
00036 *          // Keep at most 100 bytes available
00037 *          OOM_BEGIN(0.1);
00038 *          ptr=malloc(2000);
00039 *          OOM_END;
00040 *          if (NULL == ptr)
00041 *              fprintf(stderr,"malloc failed\n");
00042 *          else
00043 *              free(ptr);
00044 *      }
00045 *
00046 *      size_t ramlimit;
00047 *      ramlimit=64*1024*1024;
00048 *      printf ("Limit memory usage to %uMB : ",ramlimit/1024/1024);
00049 *      ramlimit = oomtest_setup(ramlimit);
00050 *      printf ("%uB (%uMB)\n",ramlimit,ramlimit/1024/1024);
00051 *
00052 *      // not constrained (only by the 64MB ramlimit)
00053 *      myfunction();
00054 *
00055 *      // OOM activation to eat all memory
00056 *      oomtest_enable(64*1024*1024);
00057 *      myfunction();
00058 */
00059
00090 size_t oomtest_config(const size_t hardlimit);
00091
00120 extern size_t (*oomtest_enable)(const size_t softlimit);
00121
00145 extern size_t (*oomtest_disable)();
00146
00176 extern size_t (*oomtest_fill)(const size_t minHeap, const size_t minStack);
00177
00193 extern void (*oomtest_free)();
00194
00195 extern unsigned char oomtest_enabled();
00196
00197 #endif /*__OOM_H__ */
```

5.21 gettext.h File Reference

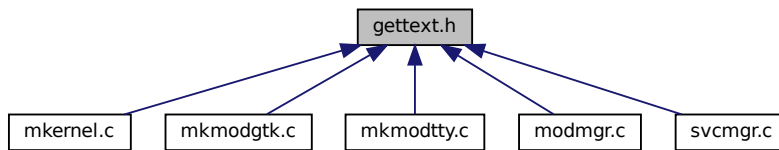
```
#include <string.h>
```

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

Include dependency graph for gettext.h:



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



Macros

- #define `gettext(Msgid)` ((const char *) (Msgid))
- #define `dgettext(Domainname, Msgid)` ((void) (Domainname), `gettext` (Msgid))
- #define `dcgettext(Domainname, Msgid, Category)` ((void) (Category), `dgettext` (Domainname, Msgid))
- #define `nggettext(Msgid1, Msgid2, N)`
- #define `dngettext(Domainname, Msgid1, Msgid2, N)` ((void) (Domainname), `nggettext` (Msgid1, Msgid2, N))
- #define `dcngettext(Domainname, Msgid1, Msgid2, N, Category)` ((void) (Category), `dngettext` (Domainname, Msgid1, Msgid2, N))
- #define `textdomain(Domainname)` ((const char *) (Domainname))
- #define `bindtextdomain(Domainname, Dirname)` ((void) (Domainname), (const char *) (Dirname))
- #define `bind_textdomain_codeset(Domainname, Codeset)` ((void) (Domainname), (const char *) (Codeset))
- #define `gettext_noop(String)` String
- #define `GETTEXT_CONTEXT_GLUE` "\004"
- #define `pgettext(Msgctxt, Msgid)` `pgettext_aux` (`NULL`, Msgctxt `GETTEXT_CONTEXT_GLUE` Msgid, Msgid, LC_MESSAGES)
- #define `dpgettext(Domainname, Msgctxt, Msgid)` `pgettext_aux` (Domainname, Msgctxt `GETTEXT_CONTEXT_GLUE` Msgid, Msgid, LC_MESSAGES)
- #define `dcpgettext(Domainname, Msgctxt, Msgid, Category)` `pgettext_aux` (Domainname, Msgctxt `GETTEXT_CONTEXT_GLUE` Msgid, Msgid, Category)
- #define `npgettext(Msgctxt, Msgid, MsgidPlural, N)` `npgettext_aux` (`NULL`, Msgctxt `GETTEXT_CONTEXT_GLUE` Msgid, Msgid, MsgidPlural, N, LC_MESSAGES)
- #define `dngettext(Domainname, Msgctxt, Msgid, MsgidPlural, N)` `npgettext_aux` (Domainname, Msgctxt `GETTEXT_CONTEXT_GLUE` Msgid, Msgid, MsgidPlural, N, LC_MESSAGES)
- #define `dcnpgettext(Domainname, Msgctxt, Msgid, MsgidPlural, N, Category)` `npgettext_aux` (Domainname, Msgctxt `GETTEXT_CONTEXT_GLUE` Msgid, Msgid, MsgidPlural, N, Category)
- #define `_LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS` 0
- #define `pgettext_expr(Msgctxt, Msgid)` `dcpgettext_expr` (`NULL`, Msgctxt, Msgid, LC_MESSAGES)
- #define `dpgettext_expr(Domainname, Msgctxt, Msgid)` `dcpgettext_expr` (Domainname, Msgctxt, Msgid, LC_MESSAGES)
- #define `npgettext_expr(Msgctxt, Msgid, MsgidPlural, N)` `dcnpgettext_expr` (`NULL`, Msgctxt, Msgid, MsgidPlural, N, LC_MESSAGES)
- #define `dngettext_expr(Domainname, Msgctxt, Msgid, MsgidPlural, N)` `dcnpgettext_expr` (Domainname, Msgctxt, Msgid, MsgidPlural, N, LC_MESSAGES)

5.21.1 Macro Definition Documentation

5.21.1.1 `_LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS`

```
#define _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS 0
```

Definition at line 190 of file [gettext.h](#).

5.21.1.2 `bind_textdomain_codeset`

```
#define bind_textdomain_codeset(  
    Domainname,  
    Codeset ) ((void) (Domainname), (const char *) (Codeset))
```

Definition at line 90 of file [gettext.h](#).

5.21.1.3 `bindtextdomain`

```
#define bindtextdomain(  
    Domainname,  
    Dirname ) ((void) (Domainname), (const char *) (Dirname))
```

Definition at line 87 of file [gettext.h](#).

5.21.1.4 `dcgettext`

```
#define dcgettext(  
    Domainname,  
    Msgid,  
    Category ) ((void) (Category), dgettext (Domainname, Msgid))
```

Definition at line 71 of file [gettext.h](#).

5.21.1.5 `dcngettext`

```
#define dcngettext(  
    Domainname,  
    Msgid1,  
    Msgid2,  
    N,  
    Category ) ((void) (Category), dngettext (Domainname, Msgid1, Msgid2, N))
```

Definition at line 82 of file [gettext.h](#).

5.21.1.6 dcngettext

```
#define dcngettext(  
    Domainname,  
    Msgctxt,  
    Msgid,  
    MsgidPlural,  
    N,  
    Category ) npgettext_aux (Domainname, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid,  
MsgidPlural, N, Category)
```

Definition at line 137 of file [gettext.h](#).

5.21.1.7 dcpgettext

```
#define dcpgettext(  
    Domainname,  
    Msgctxt,  
    Msgid,  
    Category ) pgettext_aux (Domainname, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid,  
Category)
```

Definition at line 126 of file [gettext.h](#).

5.21.1.8 dgettext

```
#define dgettext(  
    Domainname,  
    Msgid ) ((void) (Domainname), gettext (Msgid))
```

Definition at line 69 of file [gettext.h](#).

5.21.1.9 dngettext

```
#define dngettext(  
    Domainname,  
    Msgid1,  
    Msgid2,  
    N ) ((void) (Domainname), ngettext (Msgid1, Msgid2, N))
```

Definition at line 79 of file [gettext.h](#).

5.21.1.10 dnpgettext

```
#define dnpgettext(  
    Domainname,  
    Msgctxt,  
    Msgid,  
    MsgidPlural,  
    N) npgettext_aux (Domainname, Msgctxt GETTEXT\_CONTEXT\_GLUE Msgid, Msgid, Msgid←  
Plural, N, LC_MESSAGES)
```

Definition at line 135 of file [gettext.h](#).

5.21.1.11 dnpgettext_expr

```
#define dnpgettext_expr(  
    Domainname,  
    Msgctxt,  
    Msgid,  
    MsgidPlural,  
    N) dcpgettext_expr (Domainname, Msgctxt, Msgid, MsgidPlural, N, LC_MESSAGES)
```

Definition at line 246 of file [gettext.h](#).

5.21.1.12 dpgettext

```
#define dpgettext(  
    Domainname,  
    Msgctxt,  
    Msgid) pgettext_aux (Domainname, Msgctxt GETTEXT\_CONTEXT\_GLUE Msgid, Msgid,  
LC_MESSAGES)
```

Definition at line 124 of file [gettext.h](#).

5.21.1.13 dpgettext_expr

```
#define dpgettext_expr(  
    Domainname,  
    Msgctxt,  
    Msgid) dcpgettext_expr (Domainname, Msgctxt, Msgid, LC_MESSAGES)
```

Definition at line 199 of file [gettext.h](#).

5.21.1.14 gettext

```
#define gettext(
    Msgid ) ((const char *) (Msgid))
```

Definition at line 67 of file [gettext.h](#).

5.21.1.15 GETTEXT_CONTEXT_GLUE

```
#define GETTEXT_CONTEXT_GLUE "\004"
```

Definition at line 111 of file [gettext.h](#).

5.21.1.16 gettext_noop

```
#define gettext_noop(
    String ) String
```

Definition at line 108 of file [gettext.h](#).

5.21.1.17 ngettext

```
#define ngettext(
    Msgid1,
    Msgid2,
    N )
```

Value:

```
((N) == 1 \
 ? ((void) (Msgid2), (const char *) (Msgid1)) \
 : ((void) (Msgid1), (const char *) (Msgid2)))
```

Definition at line 74 of file [gettext.h](#).

5.21.1.18 npgettext

```
#define npgettext(
    Msgctxt,
    Msgid,
    MsgidPlural,
    N ) npgettext_aux (NULL, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, Msgid↔
Plural, N, LC_MESSAGES)
```

Definition at line 132 of file [gettext.h](#).

5.21.1.19 npgettext_expr

```
#define npgettext_expr(  
    Msgctxt,  
    Msgid,  
    MsgidPlural,  
    N ) dcngettext_expr (NULL, Msgctxt, Msgid, MsgidPlural, N, LC_MESSAGES)
```

Definition at line 244 of file [gettext.h](#).

5.21.1.20 pgettext

```
#define pgettext(  
    Msgctxt,  
    Msgid ) pgettext_aux (NULL, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, LC_↔  
MESSAGES)
```

Definition at line 121 of file [gettext.h](#).

5.21.1.21 pgettext_expr

```
#define pgettext_expr(  
    Msgctxt,  
    Msgid ) dcpgettext_expr (NULL, Msgctxt, Msgid, LC_MESSAGES)
```

Definition at line 197 of file [gettext.h](#).

5.21.1.22 textdomain

```
#define textdomain(  
    Domainname ) ((const char *) (Domainname))
```

Definition at line 85 of file [gettext.h](#).

5.22 gettext.h

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

```

00001 /* Convenience header for conditional use of GNU <libintl.h>.
00002 Copyright (C) 1995-1998, 2000-2002, 2004-2006, 2009-2016 Free Software
00003 Foundation, Inc.
00004
00005 This program is free software: you can redistribute it and/or modify
00006 it under the terms of the GNU General Public License as published by
00007 the Free Software Foundation; either version 3 of the License, or
00008 (at your option) any later version.
00009
00010 This program is distributed in the hope that it will be useful,
00011 but WITHOUT ANY WARRANTY; without even the implied warranty of
00012 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00013 GNU General Public License for more details.
00014
00015 You should have received a copy of the GNU General Public License
00016 along with this program. If not, see <http://www.gnu.org/licenses/>. */
00017
00018 #ifndef _LIBGETTEXT_H
00019 #define _LIBGETTEXT_H 1
00020
00021 /* NLS can be disabled through the configure --disable-nls option. */
00022 #if ENABLE_NLS
00023
00024 /* Get declarations of GNU message catalog functions. */
00025 # include <libintl.h>
00026
00027 /* You can set the DEFAULT_TEXT_DOMAIN macro to specify the domain used by
00028 the gettext() and ngettext() macros. This is an alternative to calling
00029 textdomain(), and is useful for libraries. */
00030 # ifdef DEFAULT_TEXT_DOMAIN
00031 #  undef gettext
00032 #  define gettext(Msgid) \
00033 dgettext (DEFAULT_TEXT_DOMAIN, Msgid)
00034 #  undef ngettext
00035 #  define ngettext(Msgid1, Msgid2, N) \
00036 dngettext (DEFAULT_TEXT_DOMAIN, Msgid1, Msgid2, N)
00037 # endif
00038
00039 #else
00040
00041 /* Solaris /usr/include/locale.h includes /usr/include/libintl.h, which
00042 chokes if dcgettext is defined as a macro. So include it now, to make
00043 later inclusions of <locale.h> a NOP. We don't include <libintl.h>
00044 as well because people using "gettext.h" will not include <libintl.h>,
00045 and also including <libintl.h> would fail on SunOS 4, whereas <locale.h>
00046 is OK. */
00047 #if defined(__sun)
00048 # include <locale.h>
00049 #endif
00050
00051 /* Many header files from the libstdc++ coming with g++ 3.3 or newer include
00052 <libintl.h>, which chokes if dcgettext is defined as a macro. So include
00053 it now, to make later inclusions of <libintl.h> a NOP. */
00054 #if defined(__cplusplus) && defined(__GNUCC__) && (__GNUC__ >= 3)
00055 # include <cstdlib>
00056 # if (__GLIBC__ >= 2 && !defined __UCLIBC__) || _GLIBCXX_HAVE_LIBINTL_H
00057 #  include <libintl.h>
00058 # endif
00059 #endif
00060
00061 /* Disabled NLS.
00062 The casts to 'const char *' serve the purpose of producing warnings
00063 for invalid uses of the value returned from these functions.
00064 On pre-ANSI systems without 'const', the config.h file is supposed to
00065 contain "#define const". */
00066 # undef gettext
00067 # define gettext(Msgid) ((const char *) (Msgid))
00068 # undef dgettext
00069 # define dgettext(Domainname, Msgid) ((void) (Domainname), gettext (Msgid))
00070 # undef dcgettext
00071 # define dcgettext(Domainname, Msgid, Category) \
00072 ((void) (Category), dgettext (Domainname, Msgid))
00073 # undef ngettext
00074 # define ngettext(Msgid1, Msgid2, N) \
00075 ((N) == 1 \
00076 ? ((void) (Msgid2), (const char *) (Msgid1)) \
00077 : ((void) (Msgid1), (const char *) (Msgid2)))
00078 # undef dngettext
00079 # define dngettext(Domainname, Msgid1, Msgid2, N) \
00080 ((void) (Domainname), ngettext (Msgid1, Msgid2, N))
00081 # undef dcngettext
00082 # define dcngettext(Domainname, Msgid1, Msgid2, N, Category) \

```

```

00083 ((void) (Category), dngettext (Domainname, Msgid1, Msgid2, N))
00084 # undef textdomain
00085 # define textdomain(Domainname) ((const char *) (Domainname))
00086 # undef bindtextdomain
00087 # define bindtextdomain(Domainname, Dirname) \
00088 ((void) (Domainname), (const char *) (Dirname))
00089 # undef bind_textdomain_codeset
00090 # define bind_textdomain_codeset(Domainname, Codeset) \
00091 ((void) (Domainname), (const char *) (Codeset))
00092
00093 #endif
00094
00095 /* Prefer gnuilib's setlocale override over libintl's setlocale override.  */
00096 #ifdef GNULIB_defined_setlocale
00097 # undef setlocale
00098 # define setlocale rpl_setlocale
00099 #endif
00100
00101 /* A pseudo function call that serves as a marker for the automated
00102 extraction of messages, but does not call gettext().  The run-time
00103 translation is done at a different place in the code.
00104 The argument, String, should be a literal string.  Concatenated strings
00105 and other string expressions won't work.
00106 The macro's expansion is not parenthesized, so that it is suitable as
00107 initializer for static 'char[]' or 'const char[]' variables.  */
00108 #define gettext_noop(String) String
00109
00110 /* The separator between msgctxt and msgid in a .mo file.  */
00111 #define GETTEXT_CONTEXT_GLUE "\004"
00112
00113 /* Pseudo function calls, taking a MSGTXT and a MSGID instead of just a
00114 MSGID.  MSGTXT and MSGID must be string literals.  MSGTXT should be
00115 short and rarely need to change.
00116 The letter 'p' stands for 'particular' or 'special'.  */
00117 #ifdef DEFAULT_TEXT_DOMAIN
00118 # define pgettext(Msgctxt, Msgid) \
00119 pgettext_aux (DEFAULT_TEXT_DOMAIN, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, LC_MESSAGES)
00120 #else
00121 # define pgettext(Msgctxt, Msgid) \
00122 pgettext_aux (NULL, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, LC_MESSAGES)
00123 #endif
00124 #define dpgettext(Domainname, Msgctxt, Msgid) \
00125 pgettext_aux (Domainname, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, LC_MESSAGES)
00126 #define dcpgettext(Domainname, Msgctxt, Msgid, Category) \
00127 pgettext_aux (Domainname, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, Category)
00128 #ifdef DEFAULT_TEXT_DOMAIN
00129 # define npgettext(Msgctxt, Msgid, MsgidPlural, N) \
00130 npgettext_aux (DEFAULT_TEXT_DOMAIN, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, MsgidPlural, N,
00131 LC_MESSAGES)
00132 #else
00133 # define npgettext(Msgctxt, Msgid, MsgidPlural, N) \
00134 npgettext_aux (NULL, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, MsgidPlural, N, LC_MESSAGES)
00135 #endif
00136 #define dnpgettext(Domainname, Msgctxt, Msgid, MsgidPlural, N) \
00137 npgettext_aux (Domainname, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, MsgidPlural, N, LC_MESSAGES)
00138 #define dcngettext(Domainname, Msgctxt, Msgid, MsgidPlural, N, Category) \
00139 npgettext_aux (Domainname, Msgctxt GETTEXT_CONTEXT_GLUE Msgid, Msgid, MsgidPlural, N, Category)
00140
00141 #ifdef __GNUC__
00142 # inline
00143 # else
00144 # ifdef __cplusplus
00145 inline
00146 # endif
00147 static const char *
00148 pgettext_aux (const char *domain,
00149               const char *msg_ctxt_id, const char *msgid,
00150               int category)
00151 {
00152     const char *translation = dcgettext (domain, msg_ctxt_id, category);
00153     if (translation == msg_ctxt_id)
00154         return msgid;
00155     else
00156         return translation;
00157 }
00158
00159 #ifdef __GNUC__
00160 # inline
00161 # else
00162 # ifdef __cplusplus
00163 inline
00164 # endif
00165 static const char *
00166 npgettext_aux (const char *domain,
00167                const char *msg_ctxt_id, const char *msgid,

```

```

00169         const char *msgid_plural, unsigned long int n,
00170         int category)
00171 {
00172     const char *translation =
00173         dcngettext (domain, msg_ctxt_id, msgid_plural, n, category);
00174     if (translation == msg_ctxt_id || translation == msgid_plural)
00175         return (n == 1 ? msgid : msgid_plural);
00176     else
00177         return translation;
00178 }
00179
00180 /* The same thing extended for non-constant arguments.   Here MSGTXT and MSGID
00181 can be arbitrary expressions.   But for string literals these macros are
00182 less efficient than those above.   */
00183
00184 #include <string.h>
00185
00186 #if ((__GNUC__ >= 3 || __GNUG__ >= 2) && !defined __STRICT_ANSI__) \
00187 /* || __STDC_VERSION__ >= 199901L */)
00188 # define _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS 1
00189 #else
00190 # define _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS 0
00191 #endif
00192
00193 #if !_LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS
00194 #include <stdlib.h>
00195 #endif
00196
00197 #define pgettext_expr(Msgtxt, Msgid) \
00198 dcpgettext_expr (NULL, Msgtxt, Msgid, LC_MESSAGES)
00199 #define dpgettext_expr(Domainname, Msgtxt, Msgid) \
00200 dcpgettext_expr (Domainname, Msgtxt, Msgid, LC_MESSAGES)
00201
00202 #ifdef __GNUC__
00203     __inline
00204 #else
00205 #ifdef __cplusplus
00206     inline
00207 #endif
00208 #endif
00209 static const char *
00210 dcpgettext_expr (const char *domain,
00211                 const char *msgtxt, const char *msgid,
00212                 int category)
00213 {
00214     size_t msgtxt_len = strlen (msgtxt) + 1;
00215     size_t msgid_len = strlen (msgid) + 1;
00216     const char *translation;
00217     #if _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS
00218     char msg_ctxt_id[msgtxt_len + msgid_len];
00219     #else
00220     char buf[1024];
00221     char *msg_ctxt_id =
00222         (msgtxt_len + msgid_len <= sizeof (buf)
00223          ? buf
00224          : (char *) malloc (msgtxt_len + msgid_len));
00225     if (msg_ctxt_id != NULL)
00226     #endif
00227     {
00228         int found_translation;
00229         memcpy (msg_ctxt_id, msgtxt, msgtxt_len - 1);
00230         msg_ctxt_id[msgtxt_len - 1] = '\004';
00231         memcpy (msg_ctxt_id + msgtxt_len, msgid, msgid_len);
00232         translation = dgettext (domain, msg_ctxt_id, category);
00233         found_translation = (translation != msg_ctxt_id);
00234     #if !_LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS
00235         if (msg_ctxt_id != buf)
00236             free (msg_ctxt_id);
00237     #endif
00238         if (found_translation)
00239             return translation;
00240     }
00241     return msgid;
00242 }
00243
00244 #define npgettext_expr(Msgtxt, Msgid, MsgidPlural, N) \
00245 dcngettext_expr (NULL, Msgtxt, Msgid, MsgidPlural, N, LC_MESSAGES)
00246 #define dnpgettext_expr(Domainname, Msgtxt, Msgid, MsgidPlural, N) \
00247 dcngettext_expr (Domainname, Msgtxt, Msgid, MsgidPlural, N, LC_MESSAGES)
00248
00249 #ifdef __GNUC__
00250     __inline
00251 #else
00252 #ifdef __cplusplus
00253     inline
00254 #endif
00255 #endif

```

```

00256 static const char *
00257 dcnpgettext_expr (const char *domain,
00258                  const char *msgctxt, const char *msgid,
00259                  const char *msgid_plural, unsigned long int n,
00260                  int category)
00261 {
00262     size_t msgctxt_len = strlen (msgctxt) + 1;
00263     size_t msgid_len = strlen (msgid) + 1;
00264     const char *translation;
00265     #if _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS
00266     char msg_ctxt_id[msgctxt_len + msgid_len];
00267     #else
00268     char buf[1024];
00269     char *msg_ctxt_id =
00270     (msgctxt_len + msgid_len <= sizeof (buf)
00271     ? buf
00272     : (char *) malloc (msgctxt_len + msgid_len));
00273     if (msg_ctxt_id != NULL)
00274     #endif
00275     {
00276         int found_translation;
00277         memcpy (msg_ctxt_id, msgctxt, msgctxt_len - 1);
00278         msg_ctxt_id[msgctxt_len - 1] = '\\004';
00279         memcpy (msg_ctxt_id + msgctxt_len, msgid, msgid_len);
00280         translation = dcngettext (domain, msg_ctxt_id, msgid_plural, n, category);
00281         found_translation = !(translation == msg_ctxt_id || translation == msgid_plural);
00282     #if !_LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS
00283         if (msg_ctxt_id != buf)
00284             free (msg_ctxt_id);
00285     #endif
00286         if (found_translation)
00287             return translation;
00288     }
00289     return (n == 1 ? msgid : msgid_plural);
00290 }
00291
00292 #endif /* _LIBGETTEXT_H */

```

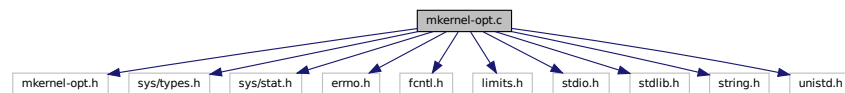
5.23 mkernel-opt.c File Reference

```

#include "mkkernel-opt.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <fcntl.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

```

Include dependency graph for mkkernel-opt.c:



Macros

- #define `OPTION_CODE_COMPILE` 1
- #define `zCopyright` (mkkernel_opt_strs+0)
- #define `zLicenseDescrip` (mkkernel_opt_strs+266)
- #define `NULL` 0

- #define `MODULE_PATH_DESC` (mkernel_opt_strs+892)
module-path option description:
- #define `MODULE_PATH_NAME` (mkernel_opt_strs+918)
Upper-cased name for the module-path option.
- #define `MODULE_PATH_name` (mkernel_opt_strs+930)
Name string for the module-path option.
- #define `MODULE_PATH_FLAGS`
Compiled in flag settings for the module-path option.
- #define `HELP_DESC` (mkernel_opt_strs+942)
- #define `HELP_name` (mkernel_opt_strs+986)
- #define `MORE_HELP_DESC HELP_DESC`
- #define `MORE_HELP_name HELP_name`
- #define `MORE_HELP_FLAGS` (OPTST_OMITTED | OPTST_NO_INIT)
- #define `VER_FLAGS`
- #define `VER_DESC` (mkernel_opt_strs+1046)
- #define `VER_name` (mkernel_opt_strs+1082)
- #define `SAVE_OPTS_DESC` (mkernel_opt_strs+1090)
- #define `SAVE_OPTS_name` (mkernel_opt_strs+1129)
- #define `LOAD_OPTS_DESC` (mkernel_opt_strs+1139)
- #define `LOAD_OPTS_NAME` (mkernel_opt_strs+1171)
- #define `NO_LOAD_OPTS_name` (mkernel_opt_strs+1181)
- #define `LOAD_OPTS_pfx` (mkernel_opt_strs+1194)
- #define `LOAD_OPTS_name` (`NO_LOAD_OPTS_name` + 3)
- #define `VER_PROC` `optionPrintVersion`
- #define `zPROGNAME` (mkernel_opt_strs+1197)
Reference to the upper cased version of mkernel.
- #define `zUsageTitle` (mkernel_opt_strs+1205)
Reference to the title line for mkernel usage.
- #define `zRcName` (mkernel_opt_strs+1321)
mkernel configuration file name.
- #define `zBugsAddr` (mkernel_opt_strs+1332)
The mkernel program bug email address.
- #define `zExplain` (mkernel_opt_strs+1354)
Clarification/explanation of what mkernel does.
- #define `zDetail` (mkernel_opt_strs+1423)
Extra detail explaining what mkernel does.
- #define `zFullVersion` (mkernel_opt_strs+1610)
The full version string for mkernel.
- #define `OPTPROC_BASE` `OPTPROC_NONE`
- #define `translate_option_strings` `NULL`
- #define `mkernel_full_usage` (`NULL`)
- #define `mkernel_short_usage` (`NULL`)
- #define `PKGDATA DIR ""`
The directory containing the data associated with mkernel.
- #define `mkernel_packager_info` `NULL`
Information about the person or institution that packaged mkernel for the current distribution.

Variables

- FILE * [option_usage_fp](#)
- tOptProc [optionBooleanVal](#)
Declare option callback procedures.
- tOptProc [optionNestedVal](#)
- tOptProc [optionNumericVal](#)
- tOptProc [optionPagedUsage](#)
- tOptProc [optionPrintVersion](#)
- tOptProc [optionResetOpt](#)
- tOptProc [optionStackArg](#)
- tOptProc [optionTimeDate](#)
- tOptProc [optionTimeVal](#)
- tOptProc [optionUnstackArg](#)
- tOptProc [optionVendorOption](#)
- tOptions [mkernelOptions](#)
The option definitions for mkernel.

5.23.1 Macro Definition Documentation

5.23.1.1 HELP_DESC

```
#define HELP_DESC (mkernel_opt_strs+942)
```

Definition at line 129 of file [mkernel-opt.c](#).

5.23.1.2 HELP_name

```
#define HELP_name (mkernel_opt_strs+986)
```

Definition at line 130 of file [mkernel-opt.c](#).

5.23.1.3 LOAD_OPTS_DESC

```
#define LOAD_OPTS_DESC (mkernel_opt_strs+1139)
```

Definition at line 150 of file [mkernel-opt.c](#).

5.23.1.4 LOAD_OPTS_NAME

```
#define LOAD_OPTS_NAME (mkernel_opt_strs+1171)
```

Definition at line 151 of file [mkernel-opt.c](#).

5.23.1.5 LOAD_OPTS_name

```
#define LOAD_OPTS_name (NO_LOAD_OPTS_name + 3)
```

Definition at line 154 of file [mkernel-opt.c](#).

5.23.1.6 LOAD_OPTS_pfx

```
#define LOAD_OPTS_pfx (mkernel_opt_strs+1194)
```

Definition at line 153 of file [mkernel-opt.c](#).

5.23.1.7 mkernel_full_usage

```
#define mkernel_full_usage (NULL)
```

Definition at line 281 of file [mkernel-opt.c](#).

5.23.1.8 mkernel_packager_info

```
#define mkernel_packager_info NULL
```

Information about the person or institution that packaged mkernel for the current distribution.

Definition at line 321 of file [mkernel-opt.c](#).

5.23.1.9 mkernel_short_usage

```
#define mkernel_short_usage (NULL)
```

Definition at line 282 of file [mkernel-opt.c](#).

5.23.1.10 MODULE_PATH_DESC

```
#define MODULE_PATH_DESC (mkkernel_opt_strs+892)
```

module-path option description:

Descriptive text for the module-path option

Definition at line 117 of file [mkkernel-opt.c](#).

5.23.1.11 MODULE_PATH_FLAGS

```
#define MODULE_PATH_FLAGS
```

Value:

```
(OPTST_DISABLED \
 | OPTST_SET_ARGTYPE(OPARG_TYPE_STRING))
```

Compiled in flag settings for the module-path option.

Definition at line 123 of file [mkkernel-opt.c](#).

5.23.1.12 MODULE_PATH_NAME

```
#define MODULE_PATH_NAME (mkkernel_opt_strs+918)
```

Upper-cased name for the module-path option.

Definition at line 119 of file [mkkernel-opt.c](#).

5.23.1.13 MODULE_PATH_name

```
#define MODULE_PATH_name (mkkernel_opt_strs+930)
```

Name string for the module-path option.

Definition at line 121 of file [mkkernel-opt.c](#).

5.23.1.14 MORE_HELP_DESC

```
#define MORE_HELP_DESC HELP_DESC
```

Definition at line 136 of file [mkkernel-opt.c](#).

5.23.1.15 MORE_HELP_FLAGS

```
#define MORE_HELP_FLAGS (OPTST_OMITTED | OPTST_NO_INIT)
```

Definition at line 138 of file [mkernel-opt.c](#).

5.23.1.16 MORE_HELP_name

```
#define MORE_HELP_name HELP_name
```

Definition at line 137 of file [mkernel-opt.c](#).

5.23.1.17 NO_LOAD_OPTS_name

```
#define NO_LOAD_OPTS_name (mkernel_opt_strs+1181)
```

Definition at line 152 of file [mkernel-opt.c](#).

5.23.1.18 NULL

```
#define NULL 0
```

Definition at line 64 of file [mkernel-opt.c](#).

5.23.1.19 OPTION_CODE_COMPILE

```
#define OPTION_CODE_COMPILE 1
```

Definition at line 42 of file [mkernel-opt.c](#).

5.23.1.20 OPTPROC_BASE

```
#define OPTPROC_BASE OPTPROC_NONE
```

Definition at line 277 of file [mkernel-opt.c](#).

5.23.1.21 PKGDATADIR

```
#define PKGDATADIR ""
```

The directory containing the data associated with mkkernel.

Definition at line 313 of file [mkkernel-opt.c](#).

5.23.1.22 SAVE_OPTS_DESC

```
#define SAVE_OPTS_DESC (mkkernel_opt_strs+1090)
```

Definition at line 148 of file [mkkernel-opt.c](#).

5.23.1.23 SAVE_OPTS_name

```
#define SAVE_OPTS_name (mkkernel_opt_strs+1129)
```

Definition at line 149 of file [mkkernel-opt.c](#).

5.23.1.24 translate_option_strings

```
#define translate_option_strings NULL
```

Definition at line 278 of file [mkkernel-opt.c](#).

5.23.1.25 VER_DESC

```
#define VER_DESC (mkkernel_opt_strs+1046)
```

Definition at line 146 of file [mkkernel-opt.c](#).

5.23.1.26 VER_FLAGS

```
#define VER_FLAGS
```

Value:

```
(OPTST_SET_ARGTYPE(OPARG_TYPE_STRING) | \
OPTST_ARG_OPTIONAL | OPTST_IMM | OPTST_NO_INIT)
```

Definition at line 143 of file [mkkernel-opt.c](#).

5.23.1.27 VER_name

```
#define VER_name (mkernel_opt_strs+1082)
```

Definition at line 147 of file [mkernel-opt.c](#).

5.23.1.28 VER_PROC

```
#define VER_PROC optionPrintVersion
```

Definition at line 165 of file [mkernel-opt.c](#).

5.23.1.29 zBugsAddr

```
#define zBugsAddr (mkernel_opt_strs+1332)
```

The mkernel program bug email address.

Definition at line 264 of file [mkernel-opt.c](#).

5.23.1.30 zCopyright

```
#define zCopyright (mkernel_opt_strs+0)
```

Definition at line 59 of file [mkernel-opt.c](#).

5.23.1.31 zDetail

```
#define zDetail (mkernel_opt_strs+1423)
```

Extra detail explaining what mkernel does.

Definition at line 268 of file [mkernel-opt.c](#).

5.23.1.32 zExplain

```
#define zExplain (mkernel_opt_strs+1354)
```

Clarification/explanation of what mkernel does.

Definition at line 266 of file [mkernel-opt.c](#).

5.23.1.33 zFullVersion

```
#define zFullVersion (mkkernel_opt_strs+1610)
```

The full version string for mkkernel.

Definition at line 270 of file [mkkernel-opt.c](#).

5.23.1.34 zLicenseDescrip

```
#define zLicenseDescrip (mkkernel_opt_strs+266)
```

Definition at line 60 of file [mkkernel-opt.c](#).

5.23.1.35 zPROGNAME

```
#define zPROGNAME (mkkernel_opt_strs+1197)
```

Reference to the upper cased version of mkkernel.

Definition at line 254 of file [mkkernel-opt.c](#).

5.23.1.36 zRcName

```
#define zRcName (mkkernel_opt_strs+1321)
```

mkkernel configuration file name.

Definition at line 258 of file [mkkernel-opt.c](#).

5.23.1.37 zUsageTitle

```
#define zUsageTitle (mkkernel_opt_strs+1205)
```

Reference to the title line for mkkernel usage.

Definition at line 256 of file [mkkernel-opt.c](#).

5.23.2 Variable Documentation

5.23.2.1 `mkernelOptions`

```
tOptions mkkernelOptions
```

The option definitions for `mkkernel`.

The one structure that binds them all.

Definition at line [343](#) of file [mkkernel-opt.c](#).

5.23.2.2 `option_usage_fp`

```
FILE* option_usage_fp [extern]
```

5.23.2.3 `optionBooleanVal`

```
tOptProc optionBooleanVal [extern]
```

Declare option callback procedures.

5.23.2.4 `optionNestedVal`

```
tOptProc optionNestedVal
```

Definition at line [159](#) of file [mkkernel-opt.c](#).

5.23.2.5 `optionNumericVal`

```
tOptProc optionNumericVal
```

Definition at line [159](#) of file [mkkernel-opt.c](#).

5.23.2.6 `optionPagedUsage`

```
tOptProc optionPagedUsage
```

Definition at line [160](#) of file [mkkernel-opt.c](#).

5.23.2.7 optionPrintVersion

```
tOptProc optionPrintVersion
```

Definition at line 160 of file [mkernel-opt.c](#).

5.23.2.8 optionResetOpt

```
tOptProc optionResetOpt
```

Definition at line 160 of file [mkernel-opt.c](#).

5.23.2.9 optionStackArg

```
tOptProc optionStackArg
```

Definition at line 161 of file [mkernel-opt.c](#).

5.23.2.10 optionTimeDate

```
tOptProc optionTimeDate
```

Definition at line 161 of file [mkernel-opt.c](#).

5.23.2.11 optionTimeVal

```
tOptProc optionTimeVal
```

Definition at line 161 of file [mkernel-opt.c](#).

5.23.2.12 optionUnstackArg

```
tOptProc optionUnstackArg
```

Definition at line 162 of file [mkernel-opt.c](#).

5.23.2.13 optionVendorOption

tOptProc optionVendorOption

Definition at line 162 of file [mkkernel-opt.c](#).

5.24 mkkernel-opt.c

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

```

00001 /*  -- buffer-read-only: t -- vi: set ro:
00002 *
00003 * DO NOT EDIT THIS FILE (mkkernel-opt.c)
00004 *
00005 * It has been AutoGen-ed
00006 * From the definitions mkkernel-opt.def
00007 * and the template file options
00008 *
00009 * Generated from AutoOpts 42:1:17 templates.
00010 *
00011 * AutoOpts is a copyrighted work. This source file is not encumbered
00012 * by AutoOpts licensing, but is provided under the licensing terms chosen
00013 * by the mkkernel author or copyright holder. AutoOpts is
00014 * licensed under the terms of the LGPL. The redistributable library
00015 * ("libopts") is licensed under the terms of either the LGPL or, at the
00016 * users discretion, the BSD license. See the AutoOpts and/or libopts sources
00017 * for details.
00018 *
00019 * The mkkernel program is copyrighted and licensed
00020 * under the following terms:
00021 *
00022 * Copyright (C) 2017 Francois Cerbelle, all rights reserved.
00023 * This is free software. It is licensed for use, modification and
00024 * redistribution under the terms of the GNU Lesser General Public License,
00025 * version 3 or later <http://gnu.org/licenses/lgpl.html>
00026 *
00027 * mkkernel is free software: you can redistribute it and/or modify it
00028 * under the terms of the GNU Lesser General Public License as published
00029 * by the Free Software Foundation, either version 3 of the License, or
00030 * (at your option) any later version.
00031 *
00032 * mkkernel is distributed in the hope that it will be useful, but
00033 * WITHOUT ANY WARRANTY; without even the implied warranty of
00034 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
00035 * See the GNU Lesser General Public License for more details.
00036 *
00037 * You should have received a copy of the GNU Lesser General Public License
00038 * along with this program. If not, see <http://www.gnu.org/licenses/>.";
00039 */
00040
00041 #ifndef __doxygen__
00042 #define OPTION_CODE_COMPILE 1
00043 #include "mkkernel-opt.h"
00044 #include <sys/types.h>
00045 #include <sys/stat.h>
00046
00047 #include <errno.h>
00048 #include <fcntl.h>
00049 #include <limits.h>
00050 #include <stdio.h>
00051 #include <stdlib.h>
00052 #include <string.h>
00053 #include <unistd.h>
00054
00055 #ifdef __cplusplus
00056 extern "C" {
00057 #endif
00058 extern FILE * option_usage_fp;
00059 #define zCopyright (mkkernel_opt_strs+0)
00060 #define zLicenseDescrip (mkkernel_opt_strs+266)
00061
00062 #ifndef NULL
00063 # define NULL 0
00064 #endif
00065
00066 static char const mkkernel_opt_strs[1624] =
00071 /* 0 */ "mkkernel 0.0.2\n"
00072 "Copyright (C) 2017 Francois Cerbelle, all rights reserved.\n"

```



```

00073         "This is free software.  It is licensed for use, modification and\n"
00074         "redistribution under the terms of the GNU Lesser General Public License,\n"
00075         "version 3 or later <http://gnu.org/licenses/lgpl.html>\n0"
00076 /*   266 */ "mkernel is free software:  you can redistribute it and/or modify it under\n"
00077         "the terms of the GNU Lesser General Public License as published by the Free\n"
00078         "Software Foundation, either version 3 of the License, or (at your option)\n"
00079         "any later version.\n\n"
00080         "mkernel is distributed in the hope that it will be useful, but WITHOUT ANY\n"
00081         "WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS\n"
00082         "FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public License for\n"
00083         "more details.\n\n"
00084         "You should have received a copy of the GNU Lesser General Public License\n"
00085         "along with this program.  If not, see <http://www.gnu.org/licenses/>.\n";\n0"
00086 /*   892 */ "path to load modules from\n0"
00087 /*   918 */ "MODULE_PATH\n0"
00088 /*   930 */ "module-path\n0"
00089 /*   942 */ "display extended usage information and exit\n0"
00090 /*   986 */ "help\n0"
00091 /*   991 */ "extended usage information passed thru pager\n0"
00092 /*  1036 */ "more-help\n0"
00093 /*  1046 */ "output version information and exit\n0"
00094 /*  1082 */ "version\n0"
00095 /*  1090 */ "save the option state to a config file\n0"
00096 /*  1129 */ "save-opts\n0"
00097 /*  1139 */ "load options from a config file\n0"
00098 /*  1171 */ "LOAD_OPTS\n0"
00099 /*  1181 */ "no-load-opts\n0"
00100 /*  1194 */ "no\n0"
00101 /*  1197 */ "MKERNEL\n0"
00102 /*  1205 */ "mkernel - Generic micro-kernel application\n"
00103         "Usage:  %s [ -<flag> [<val>] | --<name>[={| }<val>] ]... <module>\n0"
00104 /*  1315 */ "$HOME\n0"
00105 /*  1321 */ ".mkernelrc\n0"
00106 /*  1332 */ "francois@cerbelle.net\n0"
00107 /*  1354 */ "additional information given whenever the usage routine is invoked.\n0"
00108 /*  1423 */ "This string is added to the usage output when the HELP option is selected.\n"
00109         "The contents of the file 'mkernel.details' is added to the usage output\n"
00110         "when the MORE-HELP option is selected.\n0"
00111 /*  1610 */ "mkernel 0.0.2";
00112
00117 #define MODULE_PATH_DESC      (mkkernel_opt_strs+892)
00119 #define MODULE_PATH_NAME      (mkkernel_opt_strs+918)
00121 #define MODULE_PATH_name      (mkkernel_opt_strs+930)
00123 #define MODULE_PATH_FLAGS     (OPTST_DISABLED |
00124 | OPTST_SET_ARGTYPE (OPARG_TYPE_STRING))
00125
00126 /*
00127 *   Help/More_Help/Version option descriptions:
00128 */
00129 #define HELP_DESC              (mkkernel_opt_strs+942)
00130 #define HELP_name              (mkkernel_opt_strs+986)
00131 #ifdef HAVE_WORKING_FORK
00132 #define MORE_HELP_DESC         (mkkernel_opt_strs+991)
00133 #define MORE_HELP_name         (mkkernel_opt_strs+1036)
00134 #define MORE_HELP_FLAGS       (OPTST_IMM | OPTST_NO_INIT)
00135 #else
00136 #define MORE_HELP_DESC         HELP_DESC
00137 #define MORE_HELP_name         HELP_name
00138 #define MORE_HELP_FLAGS       (OPTST_OMITTED | OPTST_NO_INIT)
00139 #endif
00140 #ifndef NO_OPTIONAL_OPT_ARGS
00141 # define VER_FLAGS             (OPTST_IMM | OPTST_NO_INIT)
00142 #else
00143 # define VER_FLAGS             (OPTST_SET_ARGTYPE (OPARG_TYPE_STRING) | \
00144 OPTST_ARG_OPTIONAL | OPTST_IMM | OPTST_NO_INIT)
00145 #endif
00146 #define VER_DESC                (mkkernel_opt_strs+1046)
00147 #define VER_name                (mkkernel_opt_strs+1082)
00148 #define SAVE_OPTS_DESC         (mkkernel_opt_strs+1090)
00149 #define SAVE_OPTS_name         (mkkernel_opt_strs+1129)
00150 #define LOAD_OPTS_DESC         (mkkernel_opt_strs+1139)
00151 #define LOAD_OPTS_NAME         (mkkernel_opt_strs+1171)
00152 #define NO_LOAD_OPTS_name      (mkkernel_opt_strs+1181)
00153 #define LOAD_OPTS_pfx          (mkkernel_opt_strs+1194)
00154 #define LOAD_OPTS_name         (NO_LOAD_OPTS_name + 3)
00155 extern tOptProc
00159     optionBooleanVal,    optionNestedVal,    optionNumericVal,
00160     optionPagedUsage,    optionPrintVersion, optionResetOpt,
00161     optionStackArg,      optionTimeDate,    optionTimeVal,
00162     optionUnstackArg,    optionVendorOption;
00163 static tOptProc
00164     doUsageOpt;
00165 #define VER_PROC             optionPrintVersion
00166
00167 /* * * * * *
00173 static tOptDesc optDesc[OPTION_CT] = {
00174     { /* entry idx, value */ 0, VALUE_OPT_MODULE_PATH,

```

```

00175 /* equiv idx, value */ 0, VALUE_OPT_MODULE_PATH,
00176 /* equivalenced to */ NO_EQUIVALENT,
00177 /* min, max, act ct */ 0, 1, 0,
00178 /* opt state flags */ MODULE_PATH_FLAGS, 0,
00179 /* last opt argumnt */ { NULL }, /* --module-path */
00180 /* arg list/cookie */ NULL,
00181 /* must/cannot opts */ NULL, NULL,
00182 /* option proc */ NULL,
00183 /* desc, NAME, name */ MODULE_PATH_DESC, MODULE_PATH_NAME, MODULE_PATH_name,
00184 /* disablement strs */ NULL, NULL },
00185
00186 { /* entry idx, value */ INDEX_OPT_VERSION, VALUE_OPT_VERSION,
00187 /* equiv idx value */ NO_EQUIVALENT, VALUE_OPT_VERSION,
00188 /* equivalenced to */ NO_EQUIVALENT,
00189 /* min, max, act ct */ 0, 1, 0,
00190 /* opt state flags */ VER_FLAGS, AOUSE_VERSION,
00191 /* last opt argumnt */ { NULL },
00192 /* arg list/cookie */ NULL,
00193 /* must/cannot opts */ NULL, NULL,
00194 /* option proc */ VER_PROC,
00195 /* desc, NAME, name */ VER_DESC, NULL, VER_name,
00196 /* disablement strs */ NULL, NULL },
00197
00198
00199
00200 { /* entry idx, value */ INDEX_OPT_HELP, VALUE_OPT_HELP,
00201 /* equiv idx value */ NO_EQUIVALENT, VALUE_OPT_HELP,
00202 /* equivalenced to */ NO_EQUIVALENT,
00203 /* min, max, act ct */ 0, 1, 0,
00204 /* opt state flags */ OPTST_IMM | OPTST_NO_INIT, AOUSE_HELP,
00205 /* last opt argumnt */ { NULL },
00206 /* arg list/cookie */ NULL,
00207 /* must/cannot opts */ NULL, NULL,
00208 /* option proc */ doUsageOpt,
00209 /* desc, NAME, name */ HELP_DESC, NULL, HELP_name,
00210 /* disablement strs */ NULL, NULL },
00211
00212 { /* entry idx, value */ INDEX_OPT_MORE_HELP, VALUE_OPT_MORE_HELP,
00213 /* equiv idx value */ NO_EQUIVALENT, VALUE_OPT_MORE_HELP,
00214 /* equivalenced to */ NO_EQUIVALENT,
00215 /* min, max, act ct */ 0, 1, 0,
00216 /* opt state flags */ MORE_HELP_FLAGS, AOUSE_MORE_HELP,
00217 /* last opt argumnt */ { NULL },
00218 /* arg list/cookie */ NULL,
00219 /* must/cannot opts */ NULL, NULL,
00220 /* option proc */ optionPagedUsage,
00221 /* desc, NAME, name */ MORE_HELP_DESC, NULL, MORE_HELP_name,
00222 /* disablement strs */ NULL, NULL },
00223
00224 { /* entry idx, value */ INDEX_OPT_SAVE_OPTS, VALUE_OPT_SAVE_OPTS,
00225 /* equiv idx value */ NO_EQUIVALENT, VALUE_OPT_SAVE_OPTS,
00226 /* equivalenced to */ NO_EQUIVALENT,
00227 /* min, max, act ct */ 0, 1, 0,
00228 /* opt state flags */ OPTST_SET_ARGTYPE(OPARG_TYPE_STRING)
00229 | OPTST_ARG_OPTIONAL | OPTST_NO_INIT, AOUSE_SAVE_OPTS,
00230 /* last opt argumnt */ { NULL },
00231 /* arg list/cookie */ NULL,
00232 /* must/cannot opts */ NULL, NULL,
00233 /* option proc */ NULL,
00234 /* desc, NAME, name */ SAVE_OPTS_DESC, NULL, SAVE_OPTS_name,
00235 /* disablement strs */ NULL, NULL },
00236
00237 { /* entry idx, value */ INDEX_OPT_LOAD_OPTS, VALUE_OPT_LOAD_OPTS,
00238 /* equiv idx value */ NO_EQUIVALENT, VALUE_OPT_LOAD_OPTS,
00239 /* equivalenced to */ NO_EQUIVALENT,
00240 /* min, max, act ct */ 0, NOLIMIT, 0,
00241 /* opt state flags */ OPTST_SET_ARGTYPE(OPARG_TYPE_STRING)
00242 | OPTST_DISABLE_IMM, AOUSE_LOAD_OPTS,
00243 /* last opt argumnt */ { NULL },
00244 /* arg list/cookie */ NULL,
00245 /* must/cannot opts */ NULL, NULL,
00246 /* option proc */ optionLoadOpt,
00247 /* desc, NAME, name */ LOAD_OPTS_DESC, LOAD_OPTS_NAME, LOAD_OPTS_name,
00248 /* disablement strs */ NO_LOAD_OPTS_name, LOAD_OPTS_pfx }
00249 };
00250
00251
00252 /* * * * * *
00254 #define zPROGNAME (mkernel_opt_strs+1197)
00256 #define zUsageTitle (mkernel_opt_strs+1205)
00258 #define zRcName (mkernel_opt_strs+1321)
00260 static char const * const apzHomeList[2] = {
00261     mkernel_opt_strs+1315,
00262     NULL };
00264 #define zBugsAddr (mkernel_opt_strs+1332)
00266 #define zExplain (mkernel_opt_strs+1354)
00268 #define zDetail (mkernel_opt_strs+1423)

```

```

00270 #define zFullVersion      (mkernel_opt_strs+1610)
00271 /* extracted from optcode.tlib near line 342 */
00272
00273 #if defined(ENABLE_NLS)
00274 # define OPTPROC_BASE OPTPROC_TRANSLATE
00275 static tOptionXlateProc translate_option_strings;
00276 #else
00277 # define OPTPROC_BASE OPTPROC_NONE
00278 # define translate_option_strings NULL
00279 #endif /* ENABLE_NLS */
00280
00281 #define mkernel_full_usage (NULL)
00282 #define mkernel_short_usage (NULL)
00283
00284 #endif /* not defined __doxygen__ */
00285
00286 /*
00287 * Create the static procedure(s) declared above.
00288 */
00296 static void
00297 doUsageOpt(tOptions * opts, tOptDesc * od)
00298 {
00299     int ex_code;
00300     ex_code = MKERNEL_EXIT_SUCCESS;
00301     optionUsage(&mkernelOptions, ex_code);
00302     /* NOTREACHED */
00303     exit(MKERNEL_EXIT_FAILURE);
00304     (void)opts;
00305     (void)od;
00306 }
00307 /* extracted from optmain.tlib near line 1250 */
00308
00312 #ifndef PKGDATADIR
00313 # define PKGDATADIR ""
00314 #endif
00315
00320 #ifndef WITH_PACKAGER
00321 # define mkernel_packager_info NULL
00322 #else
00324 static char const mkernel_packager_info[] =
00325     "Packaged by " WITH_PACKAGER
00326
00327 # ifdef WITH_PACKAGER_VERSION
00328     " ("WITH_PACKAGER_VERSION)"
00329 # endif
00330
00331 # ifdef WITH_PACKAGER_BUG_REPORTS
00332     "\nReport mkernel bugs to " WITH_PACKAGER_BUG_REPORTS
00333 # endif
00334     "\n";
00335 #endif
00336 #ifndef __doxygen__
00337
00338 #endif /* __doxygen__ */
00343 tOptions mkernelOptions = {
00344     OPTIONS_STRUCT_VERSION,
00345     0, NULL, /* original argc + argv */
00346     ( OPTPROC_BASE
00347     + OPTPROC_ERRSTOP
00348     + OPTPROC_SHORTOPT
00349     + OPTPROC_LONGOPT
00350     + OPTPROC_NO_REQ_OPT
00351     + OPTPROC_ENVIRON
00352     + OPTPROC_ARGS_REQ
00353     + OPTPROC_GNUUSAGE ),
00354     0, NULL, /* current option index, current option */
00355     NULL, NULL, zPROGNAME,
00356     zRcName, zCopyright, zLicenseDescrip,
00357     zFullVersion, apzHomeList, zUsageTitle,
00358     zExplain, zDetail, optDesc,
00359     zBugsAddr, /* address to send bugs to */
00360     NULL, NULL, /* extensions/saved state */
00361     optionUsage, /* usage procedure */
00362     translate_option_strings, /* translation procedure */
00363     /*
00364     * Indexes to special options
00365     */
00366     { INDEX_OPT_MORE_HELP, /* more-help option index */
00367     INDEX_OPT_SAVE_OPTS, /* save option index */
00368     NO_EQUIVALENT, /* '-#' option index */
00369     NO_EQUIVALENT /* index of default opt */
00370     },
00371     6 /* full option count */, 1 /* user option count */,
00372     mkernel_full_usage, mkernel_short_usage,
00373     NULL, NULL,
00374     PKGDATADIR, mkernel_packager_info
00375 };

```

```

00376
00377 #if ENABLE-NLS
00383 #include <stdio.h>
00384 #include <stdlib.h>
00385 #include <string.h>
00386 #include <unistd.h>
00387 #ifdef HAVE_DCGETTEXT
00388 # include <gettext.h>
00389 #endif
00390 #include <autoopts/usage-txt.h>
00391
00392 static char * AO_gettext(char const * pz);
00393 static void  coerce_it(void ** s);
00394
00405 static char *
00406 AO_gettext(char const * pz)
00407 {
00408     char * res;
00409     if (pz == NULL)
00410         return NULL;
00411 #ifdef HAVE_DCGETTEXT
00412     /*
00413     * While processing the option_xlateable_txt data, try to use the
00414     * "libopts" domain.  Once we switch to the option descriptor data,
00415     * do *not* use that domain.
00416     */
00417     if (option_xlateable_txt.field_ct != 0) {
00418         res = dgettext("libopts", pz);
00419         if (res == pz)
00420             res = (char *)VOIDP(_(pz));
00421     } else
00422         res = (char *)VOIDP(_(pz));
00423 #else
00424     res = (char *)VOIDP(_(pz));
00425 #endif
00426     if (res == pz)
00427         return res;
00428     res = strdup(res);
00429     if (res == NULL) {
00430         fputs(_("No memory for duping translated strings\n"), stderr);
00431         exit(MKERNEL_EXIT_FAILURE);
00432     }
00433     return res;
00434 }
00435
00440 static void coerce_it(void ** s) { *s = AO_gettext(*s);
00441 }
00442
00447 static void
00448 translate_option_strings(void)
00449 {
00450     tOptions * const opts = &mkernelOptions;
00451
00452     /*
00453     * Guard against re-translation.  It won't work.  The strings will have
00454     * been changed by the first pass through this code.  One shot only.
00455     */
00456     if (option_xlateable_txt.field_ct != 0) {
00457         /*
00458         * Do the translations.  The first pointer follows the field count
00459         * field.  The field count field is the size of a pointer.
00460         */
00461         char ** ppz = (char**)VOIDP(&(option_xlateable_txt));
00462         int    ix = option_xlateable_txt.field_ct;
00463
00464         do {
00465             ppz++; /* skip over field_ct */
00466             *ppz = AO_gettext(*ppz);
00467         } while (--ix > 0);
00468         /* prevent re-translation and disable "libopts" domain lookup */
00469         option_xlateable_txt.field_ct = 0;
00470
00471         coerce_it(VOIDP(&(opts->pzCopyright)));
00472         coerce_it(VOIDP(&(opts->pzCopyNotice)));
00473         coerce_it(VOIDP(&(opts->pzFullVersion)));
00474         coerce_it(VOIDP(&(opts->pzUsageTitle)));
00475         coerce_it(VOIDP(&(opts->pzExplain)));
00476         coerce_it(VOIDP(&(opts->pzDetail)));
00477         {
00478             tOptDesc * od = opts->pOptDesc;
00479             for (ix = opts->optCt; ix > 0; ix--, od++)
00480                 coerce_it(VOIDP(&(od->pzText)));
00481         }
00482     }
00483 }
00484 #endif /* ENABLE-NLS */
00485

```

```

00486 #ifdef DO_NOT_COMPILE_THIS_CODE_IT_IS_FOR_GETTEXT
00488 static void bogus_function(void) {
00489     /* TRANSLATORS:
00490
00491 The following dummy function was crated solely so that xgettext can
00492 extract the correct strings.  These strings are actually referenced
00493 by a field name in the mkernelOptions structure noted in the
00494 comments below.  The literal text is defined in mkernel_opt_strs.
00495
00496 NOTE: the strings below are segmented with respect to the source string
00497 mkernel_opt_strs.  The strings above are handed off for translation
00498 at run time a paragraph at a time.  Consequently, they are presented here
00499 for translation a paragraph at a time.
00500
00501 ALSO: often the description for an option will reference another option
00502 by name.  These are set off with apostrophe quotes (I hope).  Do not
00503 translate option names.
00504 */
00505     /* referenced via mkernelOptions.pzCopyright */
00506     puts(_("mkernel 0.0.2\n\
00507 Copyright (C) 2017 Francois Cerbelle, all rights reserved.\n\
00508 This is free software.  It is licensed for use, modification and\n\
00509 redistribution under the terms of the GNU Lesser General Public License,\n\
00510 version 3 or later <http://gnu.org/licenses/lgpl.html>\n"));
00511
00512     /* referenced via mkernelOptions.pzCopyNotice */
00513     puts(_("mkernel is free software: you can redistribute it and/or modify it under\n\
00514 the terms of the GNU Lesser General Public License as published by the Free\n\
00515 Software Foundation, either version 3 of the License, or (at your option)\n\
00516 any later version.\n\n"));
00517     puts(_("mkernel is distributed in the hope that it will be useful, but WITHOUT ANY\n\
00518 WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS\n\
00519 FOR A PARTICULAR PURPOSE.  See the GNU Lesser General Public License for\n\
00520 more details.\n\n"));
00521     puts(_("You should have received a copy of the GNU Lesser General Public License\n\
00522 along with this program.  If not, see <http://www.gnu.org/licenses/>.\n");\n"));
00523
00524     /* referenced via mkernelOptions.pOptDesc->pzText */
00525     puts(_("path to load modules from"));
00526
00527     /* referenced via mkernelOptions.pOptDesc->pzText */
00528     puts(_("display extended usage information and exit"));
00529
00530     /* referenced via mkernelOptions.pOptDesc->pzText */
00531     puts(_("extended usage information passed thru pager"));
00532
00533     /* referenced via mkernelOptions.pOptDesc->pzText */
00534     puts(_("output version information and exit"));
00535
00536     /* referenced via mkernelOptions.pOptDesc->pzText */
00537     puts(_("save the option state to a config file"));
00538
00539     /* referenced via mkernelOptions.pOptDesc->pzText */
00540     puts(_("load options from a config file"));
00541
00542     /* referenced via mkernelOptions.pzUsageTitle */
00543     puts(_("mkernel - Generic micro-kernel application\n\
00544 Usage:  %s [ -<flag> [<val> ] | --<name>[={| }<val> ]... <module>\n"));
00545
00546     /* referenced via mkernelOptions.pzExplain */
00547     puts(_("additional information given whenever the usage routine is invoked.\n"));
00548
00549     /* referenced via mkernelOptions.pzDetail */
00550     puts(_("This string is added to the usage output when the HELP option is selected.\n\
00551 The contents of the file 'mkernel.details' is added to the usage output\n\
00552 when the MORE-HELP option is selected.\n"));
00553
00554     /* referenced via mkernelOptions.pzFullVersion */
00555     puts(_("mkernel 0.0.2"));
00556
00557     /* referenced via mkernelOptions.pzFullUsage */
00558     puts(_("«<NOT-FOUND>»"));
00559
00560     /* referenced via mkernelOptions.pzShortUsage */
00561     puts(_("«<NOT-FOUND>»"));
00562     /* LIBOPTS-MESSAGES: */
00563     #line 67 "../autoopts.c"
00564     puts(_("allocation of %d bytes failed\n"));
00565     #line 89 "../autoopts.c"
00566     puts(_("allocation of %d bytes failed\n"));
00567     #line 48 "../init.c"
00568     puts(_("AutoOpts function called without option descriptor\n"));
00569     #line 81 "../init.c"
00570     puts(_("\\tThis exceeds the compiled library version:  "));
00571     #line 79 "../init.c"
00572     puts(_("Automated Options Processing Error!\n"
00573          "\t%s called AutoOpts function with structure version %d:%d:%d.\n"));

```

```

00574 #line 78 "../autoopts.c"
00575 puts(_("realloc of %d bytes at 0x%p failed\n"));
00576 #line 83 "../init.c"
00577 puts(_("\\tThis is less than the minimum library version:  "));
00578 #line 121 "../version.c"
00579 puts(_("Automated Options version %s\n"
00580 "\\tCopyright (C) 1999-2017 by Bruce Korb - all rights reserved\n"));
00581 #line 49 "../makeshell.c"
00582 puts(_(" (AutoOpts bug):  %s.\n"));
00583 #line 90 "../reset.c"
00584 puts(_("optionResetOpt() called, but reset-option not configured"));
00585 #line 241 "../usage.c"
00586 puts(_("could not locate the 'help' option"));
00587 #line 330 "../autoopts.c"
00588 puts(_("optionProcess() was called with invalid data"));
00589 #line 697 "../usage.c"
00590 puts(_("invalid argument type specified"));
00591 #line 568 "../find.c"
00592 puts(_("defaulted to option with optional arg"));
00593 #line 76 "../alias.c"
00594 puts(_("aliasing option is out of range.));
00595 #line 210 "../enum.c"
00596 puts(_("%s error:  the keyword '%s' is ambiguous for %s\n"));
00597 #line 78 "../find.c"
00598 puts(_(" The following options match:\n"));
00599 #line 263 "../find.c"
00600 puts(_("%s:  ambiguous option name:  %s (matches %d options)\n"));
00601 #line 161 "../check.c"
00602 puts(_("%s:  Command line arguments required\n"));
00603 #line 43 "../alias.c"
00604 puts(_("%d %s%s options allowed\n"));
00605 #line 56 "../makeshell.c"
00606 puts(_("%s error %d (%s) calling %s for '%s'\n"));
00607 #line 268 "../makeshell.c"
00608 puts(_("interprocess pipe"));
00609 #line 171 "../version.c"
00610 puts(_("error:  version option argument '%c' invalid.  Use:\n"
00611 "\\t'v' - version only\n"
00612 "\\t'c' - version and copyright\n"
00613 "\\t'n' - version and full copyright notice\n"));
00614 #line 58 "../check.c"
00615 puts(_("%s error:  the '%s' and '%s' options conflict\n"));
00616 #line 187 "../find.c"
00617 puts(_("%s:  The '%s' option has been disabled.));
00618 #line 400 "../find.c"
00619 puts(_("%s:  The '%s' option has been disabled.));
00620 #line 38 "../alias.c"
00621 puts(_("-equivalence"));
00622 #line 439 "../find.c"
00623 puts(_("%s:  illegal option -- %c\n"));
00624 #line 110 "../reset.c"
00625 puts(_("%s:  illegal option -- %c\n"));
00626 #line 241 "../find.c"
00627 puts(_("%s:  illegal option -- %s\n"));
00628 #line 740 "../find.c"
00629 puts(_("%s:  illegal option -- %s\n"));
00630 #line 118 "../reset.c"
00631 puts(_("%s:  illegal option -- %s\n"));
00632 #line 305 "../find.c"
00633 puts(_("%s:  unknown vendor extension option -- %s\n"));
00634 #line 135 "../enum.c"
00635 puts(_(" or an integer from %d through %d\n"));
00636 #line 145 "../enum.c"
00637 puts(_(" or an integer from %d through %d\n"));
00638 #line 696 "../usage.c"
00639 puts(_("%s error:  invalid option descriptor for %s\n"));
00640 #line 1030 "../usage.c"
00641 puts(_("%s error:  invalid option descriptor for %s\n"));
00642 #line 355 "../find.c"
00643 puts(_("%s:  invalid option name:  %s\n"));
00644 #line 497 "../find.c"
00645 puts(_("%s:  The '%s' option requires an argument.\n"));
00646 #line 150 "../autoopts.c"
00647 puts(_(" (AutoOpts bug):  Equivalenced option '%s' was equivalenced to both\n"
00648 "\\t'%s' and '%s'."));
00649 #line 94 "../check.c"
00650 puts(_("%s error:  The %s option is required\n"));
00651 #line 602 "../find.c"
00652 puts(_("%s:  The '%s' option cannot have an argument.\n"));
00653 #line 151 "../check.c"
00654 puts(_("%s:  Command line arguments are not allowed.\n"));
00655 #line 568 "../save.c"
00656 puts(_("error %d (%s) creating %s\n"));
00657 #line 210 "../enum.c"
00658 puts(_("%s error:  '%s' does not match any %s keywords.\n"));
00659 #line 93 "../reset.c"
00660 puts(_("%s error:  The '%s' option requires an argument.\n"));

```

```

00661 #line 122 "../save.c"
00662 puts(_("error %d (%s) stat-ing %s\n"));
00663 #line 175 "../save.c"
00664 puts(_("error %d (%s) stat-ing %s\n"));
00665 #line 143 "../restore.c"
00666 puts(_("%s error: no saved option state\n"));
00667 #line 225 "../autoopts.c"
00668 puts(_("' %s' is not a command line option.\n"));
00669 #line 113 "../time.c"
00670 puts(_("%s error:  '%s' is not a recognizable date/time.\n"));
00671 #line 50 "../time.c"
00672 puts(_("%s error:  '%s' is not a recognizable time duration.\n"));
00673 #line 92 "../check.c"
00674 puts(_("%s error: The %s option must appear %d times.\n"));
00675 #line 165 "../numeric.c"
00676 puts(_("%s error:  '%s' is not a recognizable number.\n"));
00677 #line 176 "../enum.c"
00678 puts(_("%s error:  %s exceeds %s keyword count\n"));
00679 #line 279 "../usage.c"
00680 puts(_("Try '%s %s' for more information.\n"));
00681 #line 45 "../alias.c"
00682 puts(_("one %s%s option allowed\n"));
00683 #line 170 "../makeshell.c"
00684 puts(_("standard output"));
00685 #line 905 "../makeshell.c"
00686 puts(_("standard output"));
00687 #line 223 "../usage.c"
00688 puts(_("standard output"));
00689 #line 364 "../usage.c"
00690 puts(_("standard output"));
00691 #line 574 "../usage.c"
00692 puts(_("standard output"));
00693 #line 178 "../version.c"
00694 puts(_("standard output"));
00695 #line 223 "../usage.c"
00696 puts(_("standard error"));
00697 #line 364 "../usage.c"
00698 puts(_("standard error"));
00699 #line 574 "../usage.c"
00700 puts(_("standard error"));
00701 #line 178 "../version.c"
00702 puts(_("standard error"));
00703 #line 170 "../makeshell.c"
00704 puts(_("write"));
00705 #line 905 "../makeshell.c"
00706 puts(_("write"));
00707 #line 222 "../usage.c"
00708 puts(_("write"));
00709 #line 363 "../usage.c"
00710 puts(_("write"));
00711 #line 573 "../usage.c"
00712 puts(_("write"));
00713 #line 177 "../version.c"
00714 puts(_("write"));
00715 #line 60 "../numeric.c"
00716 puts(_("%s error:  %s option value %ld is out of range.\n"));
00717 #line 44 "../check.c"
00718 puts(_("%s error:  %s option requires the %s option\n"));
00719 #line 121 "../save.c"
00720 puts(_("%s warning:  cannot save options - %s not regular file\n"));
00721 #line 174 "../save.c"
00722 puts(_("%s warning:  cannot save options - %s not regular file\n"));
00723 #line 193 "../save.c"
00724 puts(_("%s warning:  cannot save options - %s not regular file\n"));
00725 #line 567 "../save.c"
00726 puts(_("%s warning:  cannot save options - %s not regular file\n"));
00727 /* END-LIBOPTS-MESSAGES */
00728
00729 /* USAGE-TEXT: */
00730 #line 822 "../usage.c"
00731 puts(_("\t\t\t\t- an alternate for '%s'\n"));
00732 #line 1097 "../usage.c"
00733 puts(_("Version, usage and configuration options:"));
00734 #line 873 "../usage.c"
00735 puts(_("\t\t\t\t\t- default option for unnamed options\n"));
00736 #line 786 "../usage.c"
00737 puts(_("\t\t\t\t\t- disabled as '--%s'\n"));
00738 #line 1066 "../usage.c"
00739 puts(_("--- %s-%14s %s\n"));
00740 #line 1064 "../usage.c"
00741 puts(_("This option has been disabled"));
00742 #line 813 "../usage.c"
00743 puts(_("\t\t\t\t\t- enabled by default\n"));
00744 #line 40 "../alias.c"
00745 puts(_("%s error:  only "));
00746 #line 1143 "../usage.c"
00747 puts(_(" - examining environment variables named %s_*\n"));

```

```

00748 #line 168 "../file.c"
00749 puts(_("\\t\\t\\t\\t- file must not pre-exist\\n"));
00750 #line 172 "../file.c"
00751 puts(_("\\t\\t\\t\\t- file must pre-exist\\n"));
00752 #line 329 "../usage.c"
00753 puts(_("Options are specified by doubled hyphens and their name or by a single\\n"
00754 "hyphen and the flag character.\\n"));
00755 #line 882 "../makeshell.c"
00756 puts(_("\\n"
00757 "====\\n"
00758 "This incarnation of genshell will produce\\n"
00759 "a shell script to parse the options for %s:\\n\\n"));
00760 #line 142 "../enum.c"
00761 puts(_(" or an integer mask with any of the lower %d bits set\\n"));
00762 #line 846 "../usage.c"
00763 puts(_("\\t\\t\\t\\t- is a set membership option\\n"));
00764 #line 867 "../usage.c"
00765 puts(_("\\t\\t\\t\\t- must appear between %d and %d times\\n"));
00766 #line 331 "../usage.c"
00767 puts(_("Options are specified by single or double hyphens and their name.\\n"));
00768 #line 853 "../usage.c"
00769 puts(_("\\t\\t\\t\\t- may appear multiple times\\n"));
00770 #line 840 "../usage.c"
00771 puts(_("\\t\\t\\t\\t- may not be preset\\n"));
00772 #line 1258 "../usage.c"
00773 puts(_(" Arg Option-Name Description\\n"));
00774 #line 1194 "../usage.c"
00775 puts(_(" Flg Arg Option-Name Description\\n"));
00776 #line 1252 "../usage.c"
00777 puts(_(" Flg Arg Option-Name Description\\n"));
00778 #line 1253 "../usage.c"
00779 puts(_(" %3s %s"));
00780 #line 1259 "../usage.c"
00781 puts(_(" %3s %s"));
00782 #line 336 "../usage.c"
00783 puts(_("The '-#<number>' option may omit the hash char\\n"));
00784 #line 332 "../usage.c"
00785 puts(_("All arguments are named options.\\n"));
00786 #line 920 "../usage.c"
00787 puts(_(" - reading file %s"));
00788 #line 358 "../usage.c"
00789 puts(_("\\n"
00790 "Please send bug reports to: <%s>\\n"));
00791 #line 100 "../version.c"
00792 puts(_("\\n"
00793 "Please send bug reports to: <%s>\\n"));
00794 #line 129 "../version.c"
00795 puts(_("\\n"
00796 "Please send bug reports to: <%s>\\n"));
00797 #line 852 "../usage.c"
00798 puts(_("\\t\\t\\t\\t- may NOT appear - preset only\\n"));
00799 #line 893 "../usage.c"
00800 puts(_("\\n"
00801 "The following option preset mechanisms are supported:\\n"));
00802 #line 1141 "../usage.c"
00803 puts(_("\\n"
00804 "The following option preset mechanisms are supported:\\n"));
00805 #line 631 "../usage.c"
00806 puts(_("prohibits these options:\\n"));
00807 #line 626 "../usage.c"
00808 puts(_("prohibits the option '%s'\\n"));
00809 #line 81 "../numeric.c"
00810 puts(_("%s%d to %d"));
00811 #line 79 "../numeric.c"
00812 puts(_("%sgreater than or equal to %d"));
00813 #line 75 "../numeric.c"
00814 puts(_("%s%d exactly"));
00815 #line 68 "../numeric.c"
00816 puts(_("%sit must lie in one of the ranges:\\n"));
00817 #line 68 "../numeric.c"
00818 puts(_("%sit must be in the range:\\n"));
00819 #line 88 "../numeric.c"
00820 puts(_(" , or\\n"));
00821 #line 66 "../numeric.c"
00822 puts(_("%sis scalable with a suffix: k/K/m/M/g/G/t/T\\n"));
00823 #line 77 "../numeric.c"
00824 puts(_("%sless than or equal to %d"));
00825 #line 339 "../usage.c"
00826 puts(_("Operands and options may be intermixed. They will be reordered.\\n"));
00827 #line 601 "../usage.c"
00828 puts(_("requires the option '%s'\\n"));
00829 #line 604 "../usage.c"
00830 puts(_("requires these options:\\n"));
00831 #line 1270 "../usage.c"
00832 puts(_(" Arg Option-Name Req? Description\\n"));
00833 #line 1264 "../usage.c"
00834 puts(_(" Flg Arg Option-Name Req? Description\\n"));

```



```

00835 #line 143 "../enum.c"
00836 puts(_("or you may use a numeric representation. Preceding these with a '!'\n"
00837 "will clear the bits, specifying 'none' will clear all bits, and 'all'\n"
00838 "will set them all. Multiple entries may be passed as an option\n"
00839 "argument list.\n"));
00840 #line 859 "../usage.c"
00841 puts(_("\t\t\t\t\t- may appear up to %d times\n"));
00842 #line 52 "../enum.c"
00843 puts(_("The valid \"%s\" option keywords are:\n"));
00844 #line 1101 "../usage.c"
00845 puts(_("The next option supports vendor supported extra options:"));
00846 #line 722 "../usage.c"
00847 puts(_("These additional options are:"));
00848 /* END-USAGE-TEXT */
00849 }
00850 #endif /* uncomparable code */
00851 #ifdef __cplusplus
00852 }
00853 #endif
00854 /* mkernel-opt.c ends here */

```

5.25 mkernel.c File Reference

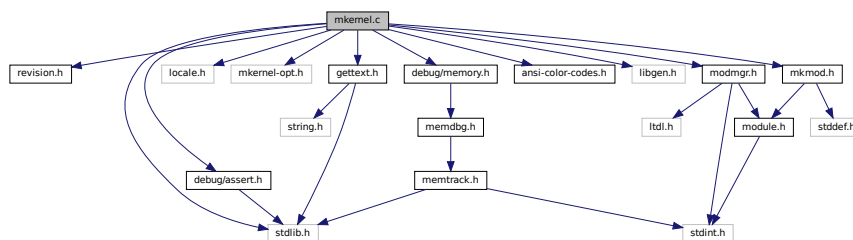
Micro-kernel core main source.

```

#include "revision.h"
#include "gettext.h"
#include <locale.h>
#include "mkernel-opt.h"
#include "modmgr.h"
#include "mkmod.h"
#include "ansi-color-codes.h"
#include <stdlib.h>
#include <libgen.h>
#include "debug/assert.h"
#include "debug/memory.h"

```

Include dependency graph for mkernel.c:



Macros

- `#define _(String) gettext (String)`
GetText helper.
- `#define PATH_MAX 255`
- `#define MODULE_PATH_ENV "MODULE_PATH"`
Default environment variable name to get module patch from.
- `#define MODULE_PATH_DEFAULT ""`
Default path to search modules in, if not defined by autotools (should be)

Functions

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

5.25.1 Detailed Description

Micro-kernel core main source.

Date

10/11/2017

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 2017, François Cerbelle

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

5.25.2 Macro Definition Documentation

5.25.2.1 `_`

```
#define _(  
    String ) gettext (String)
```

GetText helper.

Definition at line [24](#) of file [mkernel.c](#).

5.25.2.2 `MODULE_PATH_DEFAULT`

```
#define MODULE_PATH_DEFAULT "."
```

Default path to search modules in, if not defined by autotools (should be)

Definition at line [54](#) of file [mkernel.c](#).

5.25.2.3 MODULE_PATH_ENV

```
#define MODULE_PATH_ENV "MODULE_PATH"
```

Default environment variable name to get module patch from.

Todo Define in configure.ac with default value

Definition at line 49 of file [mkkernel.c](#).

5.25.2.4 PATH_MAX

```
#define PATH_MAX 255
```

Definition at line 40 of file [mkkernel.c](#).

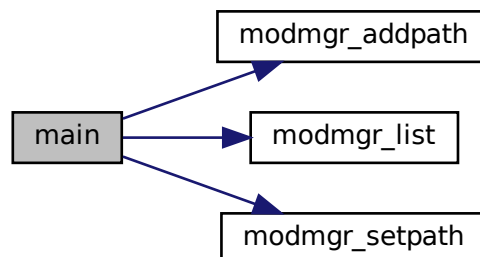
5.25.3 Function Documentation

5.25.3.1 main()

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

Definition at line 57 of file [mkkernel.c](#).

Here is the call graph for this function:



5.26 mkernel.c

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

```

00001
00017 #ifdef HAVE_CONFIG_H
00018 #include "config.h"
00019 #endif
00020
00021 #include "revision.h"
00022 #include "gettext.h"
00024 #define _(String) gettext (String)
00025 #include <locale.h>
00026
00027 /* Exclude inclusion for static code analysis with cppcheck */
00028 #ifndef NOCPPCHECK
00029 #include "mkernel-opt.h"
00030 #endif
00031
00032 #include "modmgr.h"
00033 #include "mkmod.h"
00034 #include "ansi-color-codes.h"
00035
00036 #include <stdlib.h> /* EXIT_SUCCESS / EXIT_FAILURE */
00037 #include <libgen.h> /* dirname */
00038
00039 #ifndef PATH_MAX
00040 # define PATH_MAX 255
00041 #endif
00042
00043 #include "debug/assert.h"
00044 #include "debug/memory.h"
00045
00047 #ifndef MODULE_PATH_ENV
00049 # define MODULE_PATH_ENV      "MODULE_PATH"
00050 #endif
00051
00052 #ifndef MODULE_PATH_DEFAULT
00054 # define MODULE_PATH_DEFAULT  "."
00055 #endif
00056
00057 int main(int argc, char** argv, char**env)
00058 {
00059     char *appDir;
00060
00061     (void)env;
00062
00063     /* Get executable relative invocation path from argv0 BEFORE AutoOpts */
00064     {
00065         char* execCmdCopy = strdup(argv[0]);
00066         appDir = strdup(dirname(execCmdCopy));
00067     }
00068
00069     /* LibIntl/GetText setup for Internationalization i18n */
00070     setlocale (LC_ALL, "");
00071     bindtextdomain (PACKAGE, LOCALEDIR);
00072     textdomain (PACKAGE);
00073
00074     /* Application banner */
00075     printf(BCYN PACKAGE_NAME " " PACKAGE_VERSION RESET);
00076 #ifdef REVISION
00077     printf("." BBLU REVISION RESET);
00078 #endif
00079 #ifdef BBID
00080     printf("." BBID);
00081 #endif
00082     printf("\n");
00083
00084 #pragma GCC diagnostic push /* save the actual diag context */
00085 #pragma GCC diagnostic ignored "-Wdate-time" /* locally disable warnings because of non
reproducible build triggered by pbuild */
00086     printf(_("Compiled %s at %s\n"), __DATE__, __TIME__);
00087 #pragma GCC diagnostic pop /* restore previous diag context */
00088     /* TRANSLATORS: This is a French proper name. "fraa-swa sEr\bEl" "François Cerbelle" */
00089     printf("Copyright 2024 François Cerbelle\n");
00090     printf(_("Report bugs to %s\n"), BYEL PACKAGE_BUGREPORT RESET);
00091
00092     /* AutoGen option parsing and consuming */
00093     {
00094         int arg_ct = optionProcess( &mkernelOptions, argc, argv );
00095         argc -= arg_ct;
00096         argv += arg_ct;
00097         /* Avoid assignment without usage warnings from cppcheck */
00098         (void)argv;
00099     }
00100

```

```

00101     /* Default (least significant) module search path */
00102     if (0!=modmgr_addpath(MODULE_PATH_DEFAULT))
00103         fprintf(stderr, _("The module search path can not add MODULE_PATH_DEFAULT
(%s)\n"),MODULE_PATH_DEFAULT);
00104
00105     /* Add Application root and plugins subdir to module path */
00106     {
00107         char *modulePath;
00108         modulePath = (char*)malloc(strlen(appDir)+strlen("/plugins")+1);
00109         strcpy(modulePath,appDir);
00110         strcat(modulePath,"/plugins");
00111         if (0!=modmgr_addpath(appDir))
00112             fprintf(stderr, _("The module search path can not add the application folder
(%s)\n"),appDir);
00113         if (0!=modmgr_addpath(modulePath))
00114             fprintf(stderr, _("The module search path can not add the application plugin folder
(%s)\n"),modulePath);
00115         free(modulePath);
00116     }
00117
00118     /* Override whole module path if environment defined */
00119     if ((getenv(MODULE_PATH_ENV)) && (strlen(getenv(MODULE_PATH_ENV))>0))
00120         if (0!=modmgr_setpath(getenv(MODULE_PATH_ENV)))
00121             fprintf(stderr, _("The module path can not be reset from MODULE_PATH_ENV (%s)\n"),getenv
(MODULE_PATH_ENV));
00122
00123     /* Override whole module path if options passed in CLI */
00124     if (HAVE_OPT(MODULE_PATH))
00125         if (0!=modmgr_setpath(OPT_ARG(MODULE_PATH)))
00126             fprintf(stderr, _("The module path can not be reset from CLI parameter
(%s)\n"),OPT_ARG(MODULE_PATH));
00127
00128     /* Main payload */
00129     printf(_("Hello from main\n"));
00130
00131     /* AutoGen/AutoOpts shifted left argv */
00132     /* Try to load the provided module list */
00133     {
00134         modmgr_module_t* modhandle=NULL;
00135         mkmod_api_t* *modapi=NULL;
00136         int l_i;
00137
00138         modhandle = malloc(argc*sizeof(modmgr_module_t*));
00139         memset(modhandle, 0, argc);
00140
00141         modapi = malloc(argc*sizeof(mkmod_api_t*));
00142         memset(modapi, 0, argc);
00143
00144         DBG_ITRACE(modmgr_list());
00145
00146         l_i = 0;
00147         while (l_i < argc) {
00148             DBG_PRINTF("Loading module #d: %s", l_i, argv[l_i]);
00149             MODMGR_LOAD(modhandle[l_i], modapi[l_i], argv[l_i]);
00150             if (NULL!=modhandle[l_i]) {
00151                 if ((NULL!=modapi[l_i])&&(NULL!=modapi[l_i]->mkmod_function))
00152                     modapi[l_i]->mkmod_function();
00153                 else
00154                     fprintf(stderr, _("mkmod_function not found\n"));
00155             }
00156             l_i++;
00157         }
00158
00159         DBG_ITRACE(modmgr_list());
00160
00161         l_i = 0;
00162         while (l_i < argc) {
00163             DBG_PRINTF("Unloading module #d (%s)", l_i, argv[l_i]);
00164             if (NULL!=modhandle[l_i]) {
00165                 modmgr_unload(modhandle[l_i]);
00166             }
00167             l_i++;
00168         }
00169
00170         DBG_ITRACE(modmgr_list());
00171     }
00172
00173     free(appDir);
00174     memreport();
00175
00176 #ifdef _WIN32
00177     system("PAUSE"); /* Pour la console Windows. */
00178 #endif
00179
00180     return EXIT_SUCCESS;
00181 }

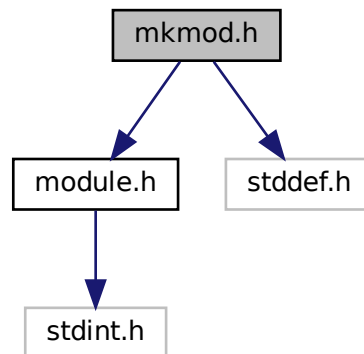
```

5.27 mkmod.h File Reference

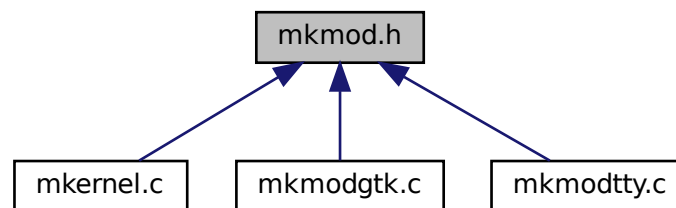
ABI interface shared between module class and application.

```
#include "module.h"  
#include <stddef.h>
```

Include dependency graph for mkmod.h:



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



Classes

- struct [mkmod_api_s](#)

Typedefs

- typedef struct [mkmod_api_s](#) [mkmod_api_t](#)

5.27.1 Detailed Description

ABI interface shared between module class and application.

Author

François Cerbelle (Fanfan), francois@cerbelle.net

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

5.27.2 Typedef Documentation

5.27.2.1 mkmod_api_t

```
typedef struct mkmod_api_s mkmod_api_t
```

5.28 mkmod.h

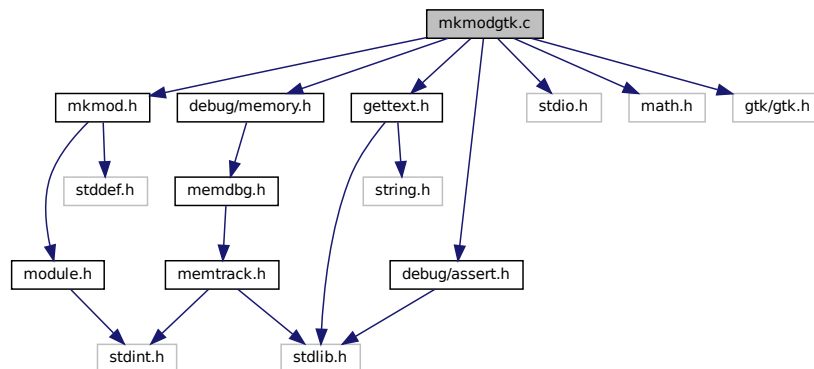
[Go to the documentation of this file.](#)

```
00001
00018 #ifndef __MKMOD_H__
00019 #define __MKMOD_H__
00020
00021 #include "module.h"
00022 # include <stddef.h> /* NULL */
00023
00024 typedef struct mkmod_api_s {
00025     void (*mkmod_function)();
00026 } mkmod_api_t;
00027
00028 #endif /* __MKMOD_H__ */
```

5.29 mkmodgtk.c File Reference

```
#include "mkmod.h"
#include "gettext.h"
#include <stdio.h>
#include "debug/assert.h"
#include "debug/memory.h"
#include <math.h>
```

```
#include <gtk/gtk.h>
Include dependency graph for mkmodgtk.c:
```



Macros

- `#define _(String) gettext (String)`

Functions

- `moduleinfo_t * onLoad ()`
- `uint8_t onUnload ()`

Variables

- `mkmod_api_t module_api`

5.29.1 Detailed Description

Date

17/11/2017

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 2017, François Cerbelle

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

5.29.2 Macro Definition Documentation

5.29.2.1 `_`

```
#define _(  
    String ) gettext (String)
```

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

5.29.3 Function Documentation

5.29.3.1 `onLoad()`

```
moduleinfo_t * onLoad ( )
```

Definition at line 90 of file [mkmodgtk.c](#).

5.29.3.2 `onUnload()`

```
uint8_t onUnload ( )
```

Definition at line 96 of file [mkmodgtk.c](#).

5.29.4 Variable Documentation

5.29.4.1 `module_api`

```
mkmod_api_t module_api
```

Initial value:

```
= {  
    mkmod_function  
}
```

Definition at line 34 of file [mkmodgtk.c](#).

5.30 mkmodgtk.c

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

```

00001
00017 #include "mkmod.h"
00018 #include "gettext.h"
00019 #define _(String) gettext (String)
00020
00021 #include <stdio.h>
00022
00023 #include "debug/assert.h"
00024 #include "debug/memory.h"
00025
00026 #include <math.h>
00027 #pragma GCC diagnostic push /* save the actual diag context */
00028 #pragma GCC diagnostic ignored "-Wpedantic" /* locally disable maybe warnings */
00029 #include <gtk/gtk.h>
00030 #pragma GCC diagnostic pop /* restore previous diag context */
00031
00032 /* List exposed module functions */
00033 static void mkmod_function();
00034 mkmod_api_t module_api = {
00035     mkmod_function
00036 };
00037
00038 static moduleinfo_t moduleinfo = {
00039     "MyGTKModule",
00040     "MyGTKModule description",
00041     0,
00042     1,
00043     0,
00044     "First and Lastname",
00045     "email@address.tld",
00046     "http://www.mygtkmodule.com",
00047     "GPLv3"
00048 };
00049
00050 static void
00051 print_hello (GtkWidget *widget,
00052             gpointer data)
00053 {
00054     (void)widget;
00055     (void)data;
00056     g_print (_("Hello world from GTK !!!"));
00057     g_print ("\n");
00058 }
00059
00060 static void
00061 activate (GtkApplication* app,
00062          gpointer user_data)
00063 {
00064     GtkWidget *window;
00065     GtkWidget *button;
00066     GtkWidget *box;
00067
00068     (void)user_data;
00069     window = gtk_application_window_new (app);
00070     gtk_window_set_title (GTK_WINDOW (window), "Window");
00071     gtk_window_set_default_size (GTK_WINDOW (window), 200, 200);
00072
00073     box = gtk_box_new (GTK_ORIENTATION_VERTICAL, 0);
00074     gtk_widget_set_halign (box, GTK_ALIGN_CENTER);
00075     gtk_widget_set_valign (box, GTK_ALIGN_CENTER);
00076
00077     gtk_window_set_child (GTK_WINDOW (window), box);
00078
00079     button = gtk_button_new_with_label (_("Hello world from GTK !!!"));
00080
00081     g_signal_connect (button, "clicked", G_CALLBACK (print_hello), NULL);
00082     g_signal_connect_swapped (button, "clicked", G_CALLBACK (gtk_window_destroy), window);
00083
00084     gtk_box_append (GTK_BOX (box), button);
00085
00086     gtk_window_present (GTK_WINDOW (window));
00087
00088 }
00089
00090 moduleinfo_t* onLoad ()
00091 {
00092     DBG_MSG("params ()");
00093     return &moduleinfo;
00094 }
00095
00096 uint8_t onUnload()
00097 {

```

```

00098     DBG_MSG("params () ");
00099     return 0;
00100 }
00101
00102 static void mkmod_function()
00103 {
00104     GtkApplication *app;
00105     DBG_MSG("params () ");
00106
00107     printf_("Hello from mkmod_function\n");
00108     app = gtk_application_new ("org.gtk.example", G_APPLICATION_DEFAULT_FLAGS);
00109     g_signal_connect (app, "activate", G_CALLBACK (activate), NULL);
00110     /* g_application_run (G_APPLICATION (app), argc, argv); */
00111     g_application_run (G_APPLICATION (app), 0, NULL);
00112     g_object_unref (app);
00113 }
00114 }
00115

```

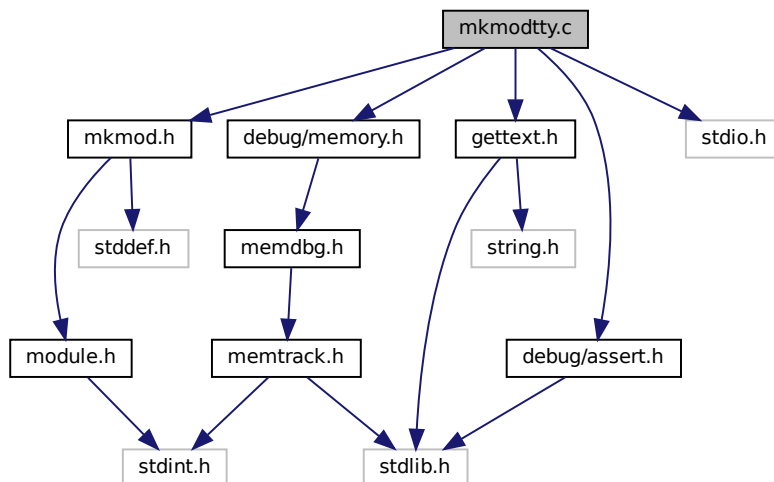
5.31 mkmodtty.c File Reference

```

#include "mkmod.h"
#include "gettext.h"
#include <stdio.h>
#include "debug/assert.h"
#include "debug/memory.h"

```

Include dependency graph for mkmodtty.c:



Macros

- `#define _(String) gettext (String)`

Functions

- `moduleinfo_t * onLoad ()`
- `uint8_t onUnload ()`

Variables

- [mkmod_api_t module_api](#)

5.31.1 Detailed Description

Date

17/11/2017

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 2017, François Cerbelle

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

5.31.2 Macro Definition Documentation

5.31.2.1 `_`

```
#define _(  
    String ) gettext (String)
```

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

5.31.3 Function Documentation

5.31.3.1 `onLoad()`

```
moduleinfo_t * onLoad ( )
```

Definition at line 43 of file [mkmodtty.c](#).

5.31.3.2 onUnload()

```
uint8_t onUnload ( )
```

Definition at line 49 of file [mkmodtty.c](#).

5.31.4 Variable Documentation

5.31.4.1 module_api

```
mkmod_api_t module_api
```

Initial value:

```
= {  
    mkmod_function  
}
```

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

5.32 mkmodtty.c

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

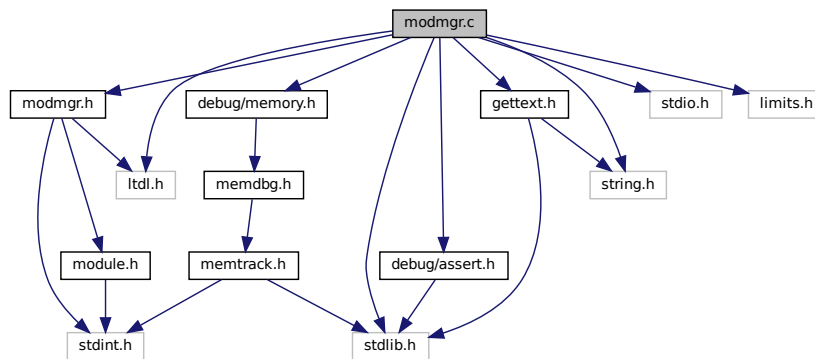
```
00001  
00017 #include "mkmod.h"  
00018 #include "gettext.h"  
00019 #define _(String) gettext (String)  
00020  
00021 #include <stdio.h>  
00022  
00023 #include "debug/assert.h"  
00024 #include "debug/memory.h"  
00025  
00026 /* List exposed module functions */static void mkmod_function();  
00027 mkmod_api_t module_api = {  
00028     mkmod_function  
00029 };  
00030  
00031 static moduleinfo_t moduleinfo = {  
00032     "MyModule",  
00033     "MyModule description",  
00034     0,  
00035     1,  
00036     0,  
00037     "First and Lastname",  
00038     "email@address.tld",  
00039     "http://www.mymodule.com",  
00040     "GPLv3"  
00041 };  
00042  
00043 moduleinfo_t* onLoad ()  
00044 {  
00045     DBG_TRACE;  
00046     return &moduleinfo;  
00047 }  
00048  
00049 uint8_t onUnload()  
00050 {  
00051     DBG_TRACE;  
00052     return 0;  
00053 }  
00054  
00055 static void mkmod_function()  
00056 {  
00057     DBG_TRACE;  
00058  
00059     printf(_("Hello from mkmod_function\n"));  
00060 }  
00061
```

5.33 modmgr.c File Reference

Module manager implementation.

```
#include "modmgr.h"
#include "gettext.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <ltdl.h>
#include "debug/assert.h"
#include "debug/memory.h"
```

Include dependency graph for modmgr.c:



Classes

- struct [modules_s](#)
Module list item structure.

Macros

- #define [_\(String\) gettext](#) (String)
- #define [PATH_MAX](#) 255

Typedefs

- typedef struct [modules_s](#) [modules_t](#)
Module list item structure.

Functions

- int [modmgr_setpath](#) (const char *path)
Reset and initialize the modules search path.
- int [modmgr_addpath](#) (const char *path)
Add a path at the end (lowest prio) of the search path.
- int [modmgr_insertpath](#) (const char *before, const char *path)
Insert (with higher prio) a path before the specified one (from the current search path)
- const char * [modmgr_getpath](#) ()
Get a read-only pointer on the current search path.
- [modmgr_module_t modmgr_load](#) (const char *modfile)
Load a module, initialize it, add it to the list and return an opaque handle.
- void [modmgr_unload](#) ([modmgr_module_t](#) module)
Decrement the usage counter, if last usage, remove from the module list and call onUnload.
- void [modmgr_list](#) ()
Output the list of modules.
- void * [modmgr_getsymbol](#) (const [modmgr_module_t](#) module, const char *szSymbol)
Resolve a module symbol pointer.

5.33.1 Detailed Description

Module manager implementation.

Date

25/11/2017

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 2017, François Cerbelle

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

5.33.2 Macro Definition Documentation

5.33.2.1 `_`

```
#define _(  
    String ) gettext (String)
```

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

5.33.2.2 PATH_MAX

```
#define PATH_MAX 255
```

Definition at line 28 of file [modmgr.c](#).

5.33.3 Typedef Documentation

5.33.3.1 modules_t

```
typedef struct modules_s modules_t
```

Module list item structure.

5.33.4 Function Documentation

5.33.4.1 modmgr_addpath()

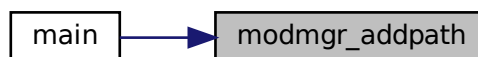
```
int modmgr_addpath (  
    const char * path )
```

Add a path at the end (lowest prio) of the search path.

Add a path to the end of the module search path.

Definition at line 141 of file [modmgr.c](#).

Here is the caller graph for this function:



5.33.4.2 modmgr_getpath()

```
const char * modmgr_getpath ( )
```

Get a read-only pointer on the current search path.

Get the current search path list.

Definition at line 181 of file [modmgr.c](#).

5.33.4.3 modmgr_getsymbol()

```
void * modmgr_getsymbol (
    const modmgr_module_t module,
    const char * szSymbol )
```

Resolve a module symbol pointer.

Resolve a module symbol, can be a function or a variable.

Definition at line 419 of file [modmgr.c](#).

5.33.4.4 modmgr_insertpath()

```
int modmgr_insertpath (
    const char * before,
    const char * path )
```

Insert (with higher prio) a path before the specified one (from the current search path)

Insert an higher priority search path before another one.

Definition at line 161 of file [modmgr.c](#).

5.33.4.5 modmgr_list()

```
void modmgr_list ( )
```

Output the list of modules.

Print the currently loaded modules list for debug and tracing.

Definition at line 379 of file [modmgr.c](#).

Here is the caller graph for this function:



5.33.4.6 modmgr_load()

```
modmgr_module_t modmgr_load (
    const char * modfile )
```

Load a module, initialize it, add it to the list and return an opaque handle.

Load a module and call the initialization with a parameter if first usage.

Todo critical section

Definition at line 191 of file [modmgr.c](#).

5.33.4.7 modmgr_setpath()

```
int modmgr_setpath (
    const char * path )
```

Reset and initialize the modules search path.

Initialize or reset module search path.

Definition at line 125 of file [modmgr.c](#).

Here is the caller graph for this function:



5.33.4.8 modmgr_unload()

```
void modmgr_unload (
    modmgr_module_t module )
```

Decrement the usage counter, if last usage, remove from the module list and call onUnload.

Call the unload function if last usage and tries to unload the module.

Definition at line 306 of file [modmgr.c](#).

5.34 modmgr.c

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

```

00001
00018 #include "modmgr.h"
00019 #include "gettext.h"
00020 #define _(String) gettext (String)
00021
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <string.h>
00025
00026 #include <limits.h>
00027 #ifndef PATH_MAX
00028 # define PATH_MAX 255
00029 #endif
00030
00031 #include <string.h>
00032 #include <lt_dl.h>
00033
00034 #if LIBPTHREAD
00035 # include <pthread.h>
00036 #endif
00037
00038 #include "debug/assert.h"
00039 #include "debug/memory.h"
00040
00042 typedef struct modules_s {
00043     lt_dlhandle handle;
00044     moduleinfo_t* modinfo;
00045     struct modules_s *next;
00046 } modules_t;
00047
00049 static modules_t* modules = NULL;
00050
00051 static char * dlerrordup (char *errmsg)
00052 {
00053     char *error = (char *) lt_dlerror ();
00054     if (error && !errmsg)
00055         errmsg = strdup (error);
00056     return errmsg;
00057 }
00058
00059 static uint8_t modmgr_error(const char* contextname)
00060 {
00061     char* errmsg=NULL;
00062
00063     errmsg = dlerrordup (errmsg);
00064     if (NULL != errmsg) {
00065         fprintf(stderr, "%s: %s\n", contextname, errmsg);
00066         free(errmsg);
00067         return 1;
00068     }
00069     DBG_PRINTF("%s: OK", contextname);
00070     return 0;
00071 }
00072
00073 static void modmgr_init_real();
00075 static void modmgr_init_noop()
00076 {
00077     DBG_MSG("modmgr already initialized.");
00078     return;
00079 }
00081 static void (*modmgr_init_ptr)() = modmgr_init_real;
00082
00084 static void modmgr_atexit()
00085 {
00086     DBG_MSG("Validate that module list is empty.");
00087     /* Checks that all modules were unloaded before exit */
00088     /* Module list should be either not initialized or with sentinel only */
00089     ASSERT((modules==NULL) || (modules->handle==NULL));
00090     /* Finished with ltdl now. */
00091     if (0!=lt_dllexit ()) {
00092         modmgr_error("lt_dllexit");
00093         /* We should abort here, but aborting in atexit() would be stupid */
00094     }
00095 }
00096
00098 static void modmgr_init_real()
00099 {
00100     /* Needs to be called once and only once */
00101     ASSERT(modules == NULL);
00102     DBG_MSG("LTDL initialization");
00103     if (0 != lt_dlinit ())

```

```

00105     modmgr_error("lt_dlinit");
00106
00107     /* Initialize the module list with a sentinel */
00108     DBG_MSG("Module list initialization");
00109     modules = (modules_t*)malloc(sizeof(modules_t));
00110     if ( NULL == modules ) {
00111         fprintf(stderr, _("Critical: Module manager lacks RAM (OOM) to initialize.\n"));
00112         abort();
00113     }
00114     modules->handle = NULL;
00115     modules->modinfo = NULL;
00116     modules->next = NULL;
00117     DBG_MSG("Register an atexit healthcheck and cleanup function");
00118     atexit(modmgr_atexit);
00119
00120     /* Avoid double calls */
00121     modmgr_init_ptr = modmgr_init_noop;
00122 }
00123
00125 int modmgr_setpath (const char* path)
00126 {
00127     int result;
00128     DBG_PRINTF("path=%s", path);
00129     modmgr_init_ptr();
00130     ASSERT(modules != NULL);
00131
00132     /* Set the module search path. */
00133     result = lt_dlsetsearchpath (path);
00134     if (0!=result)
00135         modmgr_error("lt_dlsetsearchpath");
00136     DBG_PRINTF("search path = %s", lt_dlgetsearchpath());
00137     return result;
00138 }
00139
00141 int modmgr_addpath (const char* path)
00142 {
00143     int result=0;
00144     DBG_PRINTF("path=%s", path);
00145     modmgr_init_ptr();
00146     ASSERT(modules != NULL);
00147
00148     /* Add a module search path. */
00149     if ((NULL==path)|| (0==path[0])) {
00150         fprintf(stderr, _("Module manager can not add a null or empty path.\n"));
00151     } else {
00152         result = lt_dladdsearchdir (path);
00153         if (0!=result)
00154             modmgr_error("lt_dladdsearchdir");
00155     }
00156     DBG_PRINTF("search path = %s", lt_dlgetsearchpath());
00157     return result;
00158 }
00159
00161 int modmgr_insertpath (const char* before, const char* path)
00162 {
00163     int result=0;
00164     DBG_PRINTF("before=%s, path=%s", before, path);
00165     modmgr_init_ptr();
00166     ASSERT(modules != NULL);
00167
00168     /* Insert a module search path. */
00169     if ((NULL==path)|| (0==path[0])) {
00170         fprintf(stderr, _("Module manager can not insert a null or empty path.\n"));
00171     } else {
00172         result = lt_dlinsertsearchdir (before, path);
00173         if (0!=result)
00174             modmgr_error("lt_dlinsertsearchdir");
00175     }
00176     DBG_PRINTF("search path = %s", lt_dlgetsearchpath());
00177     return result;
00178 }
00179
00181 const char* modmgr_getpath ()
00182 {
00183     DBG_TRACE;
00184     modmgr_init_ptr();
00185     ASSERT(modules != NULL);
00186
00187     return lt_dlgetsearchpath();
00188 }
00189
00191 modmgr_module_t modmgr_load(const char* modfile)
00192 {
00193     modules_t* module;
00194     moduleinfo_t* (*l_onLoad) ();
00195     modules_t* it;
00196

```

```

00197     DBG_PRINTF("modfile=%s",modfile);
00198     modmgr_init_ptr();
00199     ASSERT(modules != NULL);
00200
00201     if ((NULL==modfile)|| (0==modfile[0])) {
00202         fprintf(stderr,_"Module manager can not load a null or empty modfile.\n");
00203         return NULL;
00204     }
00205
00206     module = (modules_t*)malloc(sizeof(modules_t));
00207     if ( NULL == module ) {
00208         /* TRANSLATORS: module filename */
00209         fprintf(stderr,_"Module manager lacks RAM (OOM) to create a module structure for
%s.\n"),modfile);
00210         return NULL;
00211     }
00212
00213     DBG_PRINTF("Loading module file %s",modfile);
00214     module->handle = lt_dlopenext (modfile);
00215     if (NULL==module->handle) {
00216         modmgr_error("lt_dlopenext");
00217         fprintf(stderr,_"Module manager can not load %s.\n"),modfile);
00218         free(module);
00219         return NULL;
00220     }
00221
00222 #ifndef NDEBUG
00223     {
00224         const lt_dlinfo *li = lt_dlgetinfo(module->handle);
00225         if (0!=modmgr_error("lt_dlgetinfo")) {
00226             DBG_MSG("lt_dlgetinfo failed");
00227             abort();
00228         }
00229         DBG_PRINTF ("ltinfo:  filename = %s ", li->filename);
00230         DBG_PRINTF ("ltinfo:  name(ref_count) = %s(%d)", li->name, li->ref_count);
00231     }
00232 #endif
00233
00234     DBG_PRINTF("Search loaded module (%p) in the module list",module->handle);
00235     it = modules;
00236     while ((it)&&(it->handle!=module->handle))
00237         it = it->next;
00238
00239     /* Module already loaded and initialized, free structure and return module */
00240     if (NULL != it) {
00241         DBG_PRINTF("Module (%p) already loaded, no initialization",module->handle);
00242         free(module);
00243         return it;
00244     }
00245 #ifndef NDEBUG
00246     else
00247         DBG_PRINTF("Module (%p) not yet loaded, keep and initialize",module->handle);
00248 #endif
00249
00250     /* Find the entry points. */
00251     *(void**)(&l_onLoad) = lt_dlsym (module->handle, "onLoad");
00252
00253     /* Mandatory entry point not found, cancel the load */
00254     if (0!=modmgr_error("lt_dlsym")) {
00255         /* TRANSLATORS: module filename */
00256         fprintf(stderr,_"Module manager can not find the entry point (onLoad) not found
(%s).\n"),modfile);
00257         /* Unload to decrement the ref counter */
00258         if (0!=lt_dlclose(module->handle)) {
00259             modmgr_error("lt_dlclose");
00260             /* TRANSLATORS: module filename */
00261             fprintf(stderr,_"Critical:  Module manager can not unload partially loaded invalid module
(%s).\n"),modfile);
00262             /* Do not cleanup, immediate abort to help debugging */
00263             abort();
00264         }
00265         free(module);
00266         return NULL;
00267     }
00268
00269     /* Execute entry point to initialize, if found */
00270     DBG_PRINTF("Module (%p) entry point found (%p), initializing",module->handle, l_onLoad);
00271     module->modinfo = l_onLoad();
00272     if (module->modinfo == NULL) {
00273         fprintf(stderr,_"Invalid module(%s):  entry point (onLoad) did not return module
info.\n"),modfile);
00274         free(module);
00275         return NULL;
00276     }
00277
00278 #ifndef NDEBUG
00279     DBG_PRINTF ("modinfo:  name (version) = %s (v%d.%d.%d)",

```

```

00280         module->modinfo->moduleName,
00281         module->modinfo->moduleMajor,
00282         module->modinfo->moduleMinor,
00283         module->modinfo->modulePatch);
00284     DBG_PRINTF ("modinfo:  description = %s",
00285               module->modinfo->moduleDesc);
00286     DBG_PRINTF ("modinfo:  URL = %s",
00287               module->modinfo->moduleURL);
00288     DBG_PRINTF ("modinfo:  author (email) = %s (%s)",
00289               module->modinfo->moduleAuthor,
00290               module->modinfo->moduleEmail);
00291     DBG_PRINTF ("modinfo:  license = %s",
00292               module->modinfo->moduleLicense);
00293 #endif
00294
00295     /* Add module structure to module list */
00296     DBG_PRINTF ("Module (%p) registration in the list",module->handle);
00297     module->next = modules;
00298     modules = module;
00299
00300     DBG_PRINTF ("Return the module(%p) handle(%p)",module->handle, module);
00301     return module;
00302 }
00303
00304 void modmgr_unload(modmgr_module_t module)
00305 {
00306     modules_t *it;
00307     modules_t *prevmodule;
00308     uint8_t (*l_onUnload)();
00309
00310     DBG_PRINTF ("module(%p)",module);
00311     modmgr_init_ptr();
00312     ASSERT(modules != NULL);
00313
00314     if (NULL==module) {
00315         fprintf(stderr,_"Module manager can not unload a null module.\n");
00316         return;
00317     }
00318
00319     /* Find the module if loaded */
00320     it = modules;
00321     prevmodule = NULL;
00322     while ((it)&&(module!=it)) {
00323         prevmodule = it;
00324         it = it->next;
00325     }
00326     ASSERT(module==it); /* Module not found */
00327
00328     {
00329         /* Get the reference (loading) counter */
00330         const lt_dlinfo *li;
00331         ASSERT(NULL!=module->handle);
00332         li = lt_dlgetinfo(module->handle);
00333         if (0!=modmgr_error("lt_dlgetinfo")) {
00334             DBG_MSG("lt_dlgetinfo failed");
00335             fprintf(stderr,_"Critical:  Module manager can not get module counter before
00336 unload.\n");
00337             abort();
00338         }
00339         DBG_PRINTF ("ltinfo = %s (%s:%d)",
00340                   li->filename, li->name,
00341                   li->ref_count);
00342         /* Execute onUnLoad only if really unloading the very last occurrence */
00343         if (1==li->ref_count) {
00344             DBG_MSG("Last module usage :  call onUnLoad");
00345             *(void**)(&l_onUnload) = lt_dlsym (it->handle, "onUnLoad");
00346             if (0!=modmgr_error("lt_dlsym")) {
00347                 fprintf(stderr,_"Module manager can not find the module exit point (onUnLoad).\n");
00348             } else {
00349                 uint8_t result;
00350                 result = 0;
00351                 /* Call the exit point function.  */
00352                 result = l_onUnload();
00353                 if (result != 0)
00354                     /* TRANSLATORS:  result code, return value */
00355                     fprintf(stderr,_"Module manager received an error i(%d) from the module exit
00356 point (onUnLoad).\n"),result);
00357                 DBG_PRINTF ("onUnLoad retCode :  %d", result);
00358             }
00359
00360             /* Remove module record from module list */
00361             if (NULL!=prevmodule)
00362                 prevmodule->next = it->next;
00363             else
00364                 modules = it->next;
00365             /* Actually Unload because this was the last usage */
00366             if (0!=lt_dlclose(module->handle))

```

```

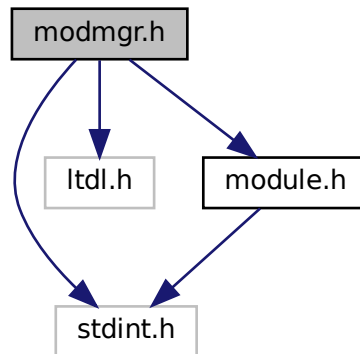
00367         modmgr_error("lt_dlclose");
00368         free(it);
00369     } else {
00370         DBG_MSG("Module still used, do not call onUnload");
00371         /* Unload : Only to decrement the ref counter */
00372         if (0!=lt_dlclose(module->handle))
00373             modmgr_error("lt_dlclose");
00374     }
00375 }
00376 }
00377
00379 void modmgr_list()
00380 {
00381     modules_t* it;
00382
00383     DBG_TRACE;
00384     modmgr_init_ptr();
00385     ASSERT(modules != NULL);
00386
00387     it = modules;
00388     printf(_("--- Module list :\n"));
00389     while (it) {
00390         /* If not on the sentinel */
00391         if (it->handle) {
00392             fprintf ( stderr,
00393                     "ltinfo = %s (%s:%d), "
00394                     "name (version) = %s (v%d.%d.%d), "
00395                     "description = %s, "
00396                     "URL = %s, "
00397                     "author (email) = %s (%s), "
00398                     "license = %s\n",
00399                     lt_dlgetinfo(it->handle)->filename,
00400                     lt_dlgetinfo(it->handle)->name,
00401                     lt_dlgetinfo(it->handle)->ref_count,
00402                     it->modinfo->moduleName,
00403                     it->modinfo->moduleMajor,
00404                     it->modinfo->moduleMinor,
00405                     it->modinfo->modulePatch,
00406                     it->modinfo->moduleDesc,
00407                     it->modinfo->moduleURL,
00408                     it->modinfo->moduleAuthor,
00409                     it->modinfo->moduleEmail,
00410                     it->modinfo->moduleLicense
00411                 );
00412         }
00413         it = it->next;
00414     }
00415     return;
00416 }
00417
00419 void* modmgr_getsymbol(const modmgr_module_t module, const char* szSymbol)
00420 {
00421     modules_t* it;
00422     void* pSymbol;
00423
00424     DBG_PRINTF("module=%p, Symbol=%s",module,szSymbol);
00425
00426     if (NULL==module) {
00427         fprintf(stderr, _("Module manager can not resolve a symbol from a null module.\n"));
00428         return NULL;
00429     }
00430
00431     if ((NULL==szSymbol)|| (0==szSymbol[0])) {
00432         fprintf(stderr, _("Module manager can not resolve a null or empty symbol name.\n"));
00433         return NULL;
00434     }
00435
00436     /* Search the module */
00437     it = modules;
00438     while ((it)&&(it!=module))
00439         it = it->next;
00440     ASSERT(NULL!=it);
00441     ASSERT(module==it);
00442
00443     pSymbol = lt_dlsym (it->handle, szSymbol);
00444     if (0!=modmgr_error("lt_dlsym")) {
00445         return NULL;
00446     }
00447
00448     return pSymbol;
00449 }

```

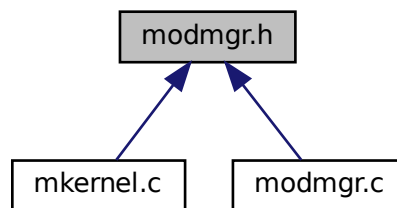
5.35 modmgr.h File Reference

Module manager headers.

```
#include <stdint.h>
#include <ltdl.h>
#include "module.h"
Include dependency graph for modmgr.h:
```



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



Macros

- #define [MODMGR_GETFUNCTION](#)(m, f) `*(void**>(&f)=modmgr_getsymbol(m,#f);`
- #define [MODMGR_LOAD](#)(module, api, filename)

Typedefs

- typedef struct [modules_s](#) * [modmgr_module_t](#)
Pointer type on private structure.

Functions

- int [modmgr_setpath](#) (const char *path)
Initialize or reset module search path.
- int [modmgr_addpath](#) (const char *path)
Add a path to the end of the module search path.
- int [modmgr_insertpath](#) (const char *before, const char *path)
Insert an higher priority search path before another one.
- const char * [modmgr_getpath](#) ()
Get the current search path list.
- [modmgr_module_t modmgr_load](#) (const char *modfile)
Load a module and call the initialization with a parameter if first usage.
- void [modmgr_unload](#) ([modmgr_module_t](#) module)
Call the unload function if last usage and tries to unload the module.
- void [modmgr_list](#) ()
Print the currently loaded modules list for debug and tracing.
- void * [modmgr_getsymbol](#) (const [modmgr_module_t](#) module, const char *szSymbol)
Resolve a module symbol, can be a function or a variable.

5.35.1 Detailed Description

Module manager headers.

Date

25/11/2017

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 2017, François Cerbelle

Manage module loading with parametrized initialization, module unloading, fetching modules ABI...

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

5.35.2 Macro Definition Documentation

5.35.2.1 MODMGR_GETFUNCTION

```
#define MODMGR_GETFUNCTION(  
    m,  
    f ) *(void**) (&f)=modmgr_getsymbol(m, #f);
```

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

5.35.2.2 MODMGR_LOAD

```
#define MODMGR_LOAD(  
    module,  
    api,  
    filename )
```

Value:

```
module=modmgr_load(filename); \  
api=modmgr_getsymbol(module,"module_api");
```

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

5.35.3 Typedef Documentation

5.35.3.1 modmgr_module_t

```
typedef struct modules_s* modmgr_module_t
```

Pointer type on private structure.

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

5.35.4 Function Documentation

5.35.4.1 modmgr_addpath()

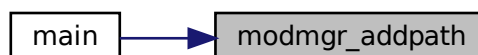
```
int modmgr_addpath (  
    const char * path )
```

Add a path to the end of the module search path.

Add a path to the end of the module search path.

Definition at line 141 of file [modmgr.c](#).

Here is the caller graph for this function:



5.35.4.2 modmgr_getpath()

```
const char * modmgr_getpath ( )
```

Get the current search path list.

Get the current search path list.

Definition at line 181 of file [modmgr.c](#).

5.35.4.3 modmgr_getsymbol()

```
void * modmgr_getsymbol (
    const modmgr_module_t module,
    const char * szSymbol )
```

Resolve a module symbol, can be a function or a variable.

Resolve a module symbol, can be a function or a variable.

Definition at line 419 of file [modmgr.c](#).

5.35.4.4 modmgr_insertpath()

```
int modmgr_insertpath (
    const char * before,
    const char * path )
```

Insert an higher priority search path before another one.

Insert an higher priority search path before another one.

Definition at line 161 of file [modmgr.c](#).

5.35.4.5 modmgr_list()

```
void modmgr_list ( )
```

Print the currently loaded modules list for debug and tracing.

Print the currently loaded modules list for debug and tracing.

Definition at line 379 of file [modmgr.c](#).

Here is the caller graph for this function:



5.35.4.6 modmgr_load()

```
modmgr_module_t modmgr_load (
    const char * modfile )
```

Load a module and call the initialization with a parameter if first usage.

Load a module and call the initialization with a parameter if first usage.

Todo critical section

Definition at line 191 of file [modmgr.c](#).

5.35.4.7 modmgr_setpath()

```
int modmgr_setpath (
    const char * path )
```

Initialize or reset module search path.

Initialize or reset module search path.

Definition at line 125 of file [modmgr.c](#).

Here is the caller graph for this function:



5.35.4.8 modmgr_unload()

```
void modmgr_unload (
    modmgr_module_t module )
```

Call the unload function if last usage and tries to unload the module.

Call the unload function if last usage and tries to unload the module.

Definition at line 306 of file [modmgr.c](#).

5.36 modmgr.h

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

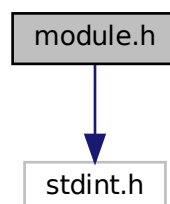
```
00001
00020 #ifndef __MODMGR_H__
00021 #define __MODMGR_H__
00022
00023 #ifdef HAVE_CONFIG_H
00024 #include "config.h"
00025 #endif
00026
00027 #include <stdint.h>
00028 #include <lttl.h>
00029
00030 #include "module.h"
00031
00032 #define MODMGR_GETFUNCTION(m, f) *(void**) (&f)=modmgr_getsymbol(m, #f);
00033 #define MODMGR_LOAD(module, api, filename) \
00034 module=modmgr_load(filename); \
00035 api=modmgr_getsymbol(module, "module_api");
00036
00037 #ifdef __cplusplus
00038 extern "C" {
00039 #endif
00040
00042 typedef struct modules_s *modmgr_module_t;
00043
00045 int modmgr_setpath (const char* path);
00046
00048 int modmgr_addpath (const char* path);
00049
00051 int modmgr_insertpath (const char* before, const char* path);
00052
00054 const char* modmgr_getpath ();
00055
00057 modmgr_module_t modmgr_load(const char* modfile);
00058
00060 void modmgr_unload(modmgr_module_t module);
00061
00063 void modmgr_list();
00064
00066 void* modmgr_getsymbol(const modmgr_module_t module, const char* szSymbol);
00067
00068 #ifdef __cplusplus
00069 }
00070 #endif
00071
00072 #endif /* __MODMGR_H__ */
```

5.37 module.h File Reference

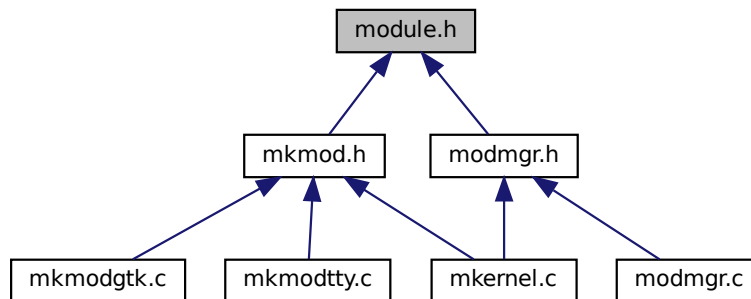
Internal ABI shared by all modules with modmgr.

```
#include <stdint.h>
```

Include dependency graph for module.h:



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



Classes

- struct [moduleinfo_s](#)

Typedefs

- typedef struct [moduleinfo_s](#) [moduleinfo_t](#)

5.37.1 Detailed Description

Internal ABI shared by all modules with modmgr.

Date

17/11/2017

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 2017, François Cerbelle

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

5.37.2 Typedef Documentation

5.37.2.1 moduleinfo_t

```
typedef struct moduleinfo_s moduleinfo_t
```

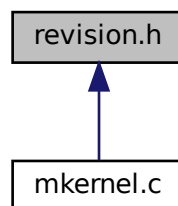
5.38 module.h

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

```
00001
00018 #ifndef __MODULE_H__
00019 #define __MODULE_H__
00020
00021 #include <stdint.h>
00022
00023 #ifdef __cplusplus
00024 extern "C" {
00025 #endif
00026
00027 typedef struct moduleinfo_s {
00028     const char* moduleName;
00029     const char* moduleDesc;
00030     const uint8_t moduleMajor;
00031     const uint8_t moduleMinor;
00032     const uint8_t modulePatch;
00033     const char* moduleAuthor;
00034     const char* moduleEmail;
00035     const char* moduleURL;
00036     const char* moduleLicense;
00037 } moduleinfo_t;
00038
00039 #ifdef __cplusplus
00040 }
00041 #endif
00042
00043 #endif /* __MODULE_H__ */
```

5.39 revision.h File Reference

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



Macros

- #define `REVISION` "e538cd7d962a"

5.39.1 Macro Definition Documentation

5.39.1.1 REVISION

```
#define REVISION "e538cd7d962a"
```

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

5.40 revision.h

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

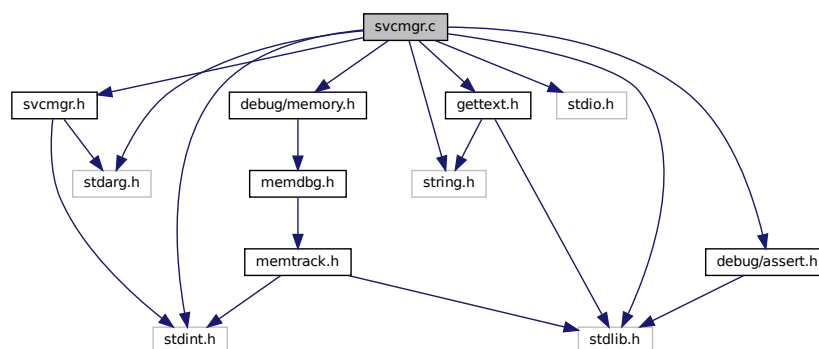
```
00001 /* This file is updated in the distdir before creating the dist archive */
00002 #ifndef REVISION
00003 #define REVISION "e538cd7d962a"
00004 #endif
```

5.41 svcmgr.c File Reference

Service manager implementation.

```
#include "svcmgr.h"
#include "gettext.h"
#include <stdarg.h>
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "debug/assert.h"
#include "debug/memory.h"
```

Include dependency graph for svcmgr.c:



Classes

- struct [service_s](#)

Macros

- #define `_(String) gettext` (String)

Typedefs

- typedef struct `service_s` `service_t`

Functions

- void `svcmgr_dump` (const char *p_prefix, `service_t` *p_list)
- void `svcmgr_register` (const char *p_endpoint, `svcfunc_t` *p_svcfunc)
- uint8_t `svcmgr_call` (const char *p_endpoint,...)
- void `svcmgr_unregister` (const char *p_endpoint)

5.41.1 Detailed Description

Service manager implementation.

Date

27/11/2017

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 2017, François Cerbelle

Todo make threadsafe

investigate prefix or b+* trees

Implements the services dictionary in an n-tree structure.

Definition in file `svcmgr.c`.

5.41.2 Macro Definition Documentation

5.41.2.1 `_`

```
#define _(  
    String ) gettext (String)
```

Definition at line 24 of file `svcmgr.c`.

5.41.3 Typedef Documentation

5.41.3.1 service_t

```
typedef struct service_s service_t
```

5.41.4 Function Documentation

5.41.4.1 svcmgr_call()

```
uint8_t svcmgr_call (
    const char * p_endpoint,
    ... )
```

Definition at line 235 of file [svcmgr.c](#).

5.41.4.2 svcmgr_dump()

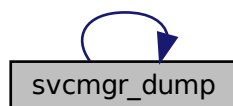
```
void svcmgr_dump (
    const char * p_prefix,
    service_t * p_list )
```

Definition at line 163 of file [svcmgr.c](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.41.4.3 svcmgr_register()

```
void svcmgr_register (
    const char * p_endpoint,
    svcfunc_t * p_svcfunc )
```

Definition at line 182 of file [svcmgr.c](#).

5.41.4.4 svcmgr_unregister()

```
void svcmgr_unregister (
    const char * p_endpoint )
```

Definition at line 259 of file [svcmgr.c](#).

5.42 svcmgr.c

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

```
00001
00022 #include "svcmgr.h"
00023 #include "gettext.h"
00024 #define _(String) gettext (String)
00025
00026 #include <stdarg.h> /* va_list, va_start, va_arg, va_end */
00027 #include <stdint.h>
00028 #include <string.h>
00029 #include <stdlib.h> /* free */
00030 #include <stdio.h> /* printf */
00031
00032 #include "debug/assert.h"
00033 #include "debug/memory.h"
00034
00035 typedef struct service_s {
00036     struct service_s *next;
00037     struct service_s *children;
00038     uint8_t nbArgs;
00039     char *name;
00040     svcfunc_t *function;
00041 } service_t;
00042
00044 static service_t* services=NULL;
00045
00052 static service_t* svcmgr_find(service_t* p_list, const char* p_name)
00053 {
00054     int l_cmp; /*< string comparison */
00055
00056     ASSERT(p_name);
00057     DBG_PRINTF("p_list=%p, p_name=%s",p_list,p_name);
00058
00059
00060     l_cmp=0;
00061     /* Search for the service name in the service list */
00062     while ((p_list!=NULL)&&((l_cmp=strcmp(p_list->name,p_name))<0)) {
00063         DBG_PRINTF("p_list=%p, p_list->next=%p",p_list,p_list->next);
00064         DBG_PRINTF("p_list=%p, p_list->name=%s",p_list,p_list->name);
00065         DBG_PRINTF("p_list=%p, l_cmp=%d",p_list,l_cmp);
00066         if (l_cmp!=0)
00067             p_list = p_list->next;
00068     }
00069
00070     return (l_cmp==0?p_list:NULL);
00071 }
00072
00083 static service_t* svcmgr_insert(service_t** p_list, service_t* p_service)
00084 {
00085     int l_cmp; /*< string comparison */
00086
00087     ASSERT(p_list);
```

```

00088     ASSERT(p_service);
00089     DBG_PRINTF("p_list=%p, p_service=%p",p_list,p_service);
00090
00091     l_cmp=0;
00092
00093     /* Search for the service name in the service list */
00094     while ((*p_list)&&((l_cmp=strcmp((*p_list)->name,p_service->name))<0))
00095         p_list = &(*p_list)->next;
00096
00097     if ((*p_list==NULL)|| (l_cmp>0)) {
00099         p_service->next = *p_list;
00100         *p_list = p_service;
00101         /* End of critical section */
00102     }
00103
00104     return *p_list;
00105 }
00106
00111 static void svcmgr_rdelete(service_t* p_service)
00112 {
00113     service_t* l_svc;
00114
00115     ASSERT(p_service);
00116     DBG_PRINTF("p_service=%p",p_service);
00117
00118     l_svc = NULL;
00119
00120     /* Delete childrens */
00121     while (p_service->children) {
00122         l_svc = p_service->children;
00123         p_service->children = p_service->next;
00124         svcmgr_rdelete(l_svc);
00125     }
00126     /* Delete name */
00127     free(p_service->name);
00128     /* Delete record */
00129     free(p_service);
00130 }
00131
00141 static service_t* svcmgr_delete(service_t** p_list, const char* p_name)
00142 {
00143     int l_cmp; /*< string comparison */
00144
00145     ASSERT(p_list);
00146     ASSERT(p_name);
00147     DBG_PRINTF("p_list=%p, p_name=%s",p_list,p_name);
00148
00149     l_cmp=0;
00150
00151     /* Search for the service name in the service list */
00152     while ((*p_list)&&((l_cmp=strcmp((*p_list)->name,p_name))<0))
00153         p_list = &(*p_list)->next;
00154
00155     if ((*p_list!=NULL)&&(l_cmp==0)) {
00156         service_t* l_svc = *p_list;
00157         *p_list = (*p_list)->next;
00158         svcmgr_rdelete(l_svc);
00159     }
00160     return *p_list;
00161 }
00162
00163 void svcmgr_dump(const char* p_prefix, service_t* p_list)
00164 {
00165     #ifndef NDEBUG
00166         while (p_list) {
00167             printf("%s%s\n",p_prefix, p_list->name);
00168             if (p_list->children) {
00169                 char l_fullname[256];
00170                 snprintf(l_fullname,255,"%s%s/",p_prefix,p_list->name);
00171                 svcmgr_dump(l_fullname,p_list->children);
00172             }
00173             p_list = p_list->next;
00174         }
00175     #else
00176         /* Avoid unused parameters compilation warning for release builds */
00177         ((void)p_prefix);
00178         ((void)p_list);
00179     #endif
00180 }
00181
00182 void svcmgr_register(const char* p_endpoint, svcfunc_t* p_svcfunc)
00183 {
00184     char* l_token;
00185     char* l_endpoint; /* Get rid of const qualifier */
00186     service_t** l_head;
00187     service_t* l_svc;
00188     service_t* l_retsvc;

```

```

00189
00190     ASSERT(p_endpoint);
00191     ASSERT(p_svcfunc);
00192     DBG_PRINTF("p_endpoint=%s, p_svcfunc=%p",p_endpoint,p_svcfunc);
00193
00194     l_endpoint = strdup(p_endpoint);
00195     DBG_PRINTF("l_endpoint=%s",l_endpoint);
00196     l_head=&services;
00197     DBG_PRINTF("l_head=%p, *l_head=%p",l_head,*l_head);
00198     l_svc=NULL;
00199     l_retsvc=NULL;
00200
00201     while ((l_token = strtok_r(l_endpoint, "/", &l_endpoint)) {
00202         DBG_PRINTF("l_token=%s",l_token);
00203         DBG_PRINTF("l_svc=%p",l_svc);
00204         l_svc = (service_t*)malloc(sizeof(service_t));
00205         DBG_PRINTF("l_svc=%p",l_svc);
00206         l_svc->next=NULL;
00207         l_svc->children=NULL;
00208         l_svc->name=strdup(l_token);
00209         if (strlen(l_endpoint)) {
00210             /* Create a intermediate node */
00211             l_svc->nbArgs=0;
00212             l_svc->function=NULL;
00213         } else {
00214             /* Create an endpoint */
00215             l_svc->nbArgs=0;
00216             l_svc->function=p_svcfunc;
00217         }
00218         DBG_MSG("-----");
00219         DBG_PRINTF("l_svc=%p",l_svc);
00220         DBG_PRINTF("l_svc->next=%p",l_svc->next);
00221         DBG_PRINTF("l_svc->children=%p",l_svc->children);
00222         DBG_PRINTF("l_svc->nbArgs=%d",l_svc->nbArgs);
00223         DBG_PRINTF("l_svc->name=%s",l_svc->name);
00224         DBG_PRINTF("l_svc->function=%p",l_svc->function);
00225         DBG_MSG("-----");
00226         if (l_svc!=(l_retsvc=svcmgr_insert(l_head, l_svc)))
00227             free(l_svc);
00228         DBG_PRINTF("l_retsvc=%p",l_retsvc);
00229         l_head=&(l_retsvc->children);
00230         DBG_PRINTF("l_head=%p, *l_head=%p",l_head,*l_head);
00231     };
00232     svcmgr_dump("/",services);
00233 }
00234
00235 uint8_t svcmgr_call(const char* p_endpoint,...)
00236 {
00237     service_t* l_svc;
00238
00239     ASSERT(p_endpoint);
00240     DBG_PRINTF("p_endpoint=%s",p_endpoint);
00241
00242     l_svc=svcmgr_find(services, p_endpoint);
00243     DBG_PRINTF("l_svc=%p",l_svc);
00244
00245     if (l_svc) {
00246         DBG_PRINTF("l_svc->function=%p",l_svc->function);
00247         if (l_svc->function) {
00248             va_list l_ap;
00249             va_start(l_ap, p_endpoint);
00250             l_svc->function(l_ap);
00251             va_end(l_ap);
00252         }
00253     }
00254
00255     DBG_TRACE;
00256     return 0;
00257 }
00258
00259 void svcmgr_unregister(const char* p_endpoint)
00260 {
00261     svcmgr_delete(&services, p_endpoint);
00262 }
00263
00264 /*
00265 sem_init
00266 sem_wait
00267 sem_post
00268 sem_destroy
00269 sem_overview
00270
00271
00272 int pthread_mutex_init(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr);
00273 int pthread_mutex_lock(pthread_mutex_t *mutex);
00274 int pthread_mutex_unlock(pthread_mutex_t *mutex);
00275 int pthread_mutex_destroy(pthread_mutex_t *mutex);

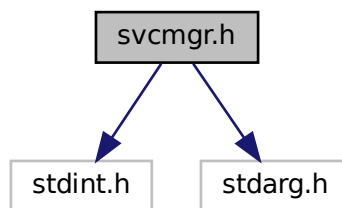
```

00276 */

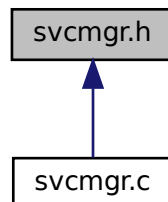
5.43 svcmgr.h File Reference

Service manager header.

```
#include <stdint.h>
#include <stdarg.h>
Include dependency graph for svcmgr.h:
```



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



Typedefs

- typedef uint8_t [svcfunc_t](#)(va_list p_ap)

Functions

- void [svcmgr_register](#) (const char *p_endpoint, [svcfunc_t](#) *p_service)
- uint8_t [svcmgr_call](#) (const char *p_endpoint,...)
- void [svcmgr_unregister](#) (const char *p_endpoint)

5.43.1 Detailed Description

Service manager header.

Date

27/11/2017

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Copyright

Copyright (c) 2017, François Cerbelle

Defines a service dictionary or registry in which a module can register and call a new service, with an endpoint, a callback function and an arbitrary number of arguments.

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

5.43.2 Typedef Documentation

5.43.2.1 svcfunc_t

```
typedef uint8_t svcfunc_t(va_list p_ap)
```

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

5.43.3 Function Documentation

5.43.3.1 svcmgr_call()

```
uint8_t svcmgr_call (  
    const char * p_endpoint,  
    ... )
```

Definition at line 235 of file [svcmgr.c](#).

5.43.3.2 svcmgr_register()

```
void svcmgr_register (
    const char * p_endpoint,
    svcfunc_t * p_service )
```

Definition at line 182 of file [svcmgr.c](#).

5.43.3.3 svcmgr_unregister()

```
void svcmgr_unregister (
    const char * p_endpoint )
```

Definition at line 259 of file [svcmgr.c](#).

5.44 svcmgr.h

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

```
00001
00020 #ifndef __SVCGR_H__
00021 #define __SVCGR_H__
00022
00023 #ifdef HAVE_CONFIG_H
00024 #include "config.h"
00025 #endif
00026
00027 #include <stdint.h>
00028 #include <stdarg.h> /* va_list, va_start, va_arg, va_end */
00029
00030 #ifdef __cplusplus
00031 extern "C" {
00032 #endif
00033
00034 typedef uint8_t svcfunc_t (va_list p_ap);
00035
00036 void svcmgr_register(const char* p_endpoint, svcfunc_t* p_service);
00037 uint8_t svcmgr_call(const char* p_endpoint, ...);
00038 void svcmgr_unregister(const char* p_endpoint);
00039
00040 #ifdef __cplusplus
00041 }
00042 #endif
00043
00044 #endif /* __SVCGR_H__ */
```


Index

- - mkkernel.c, [126](#)
 - mkmodgtk.c, [133](#)
 - mkmodtty.c, [136](#)
 - modmgr.c, [139](#)
 - svcmgr.c, [157](#)
- _GNU_SOURCE
 - oom.c, [81](#)
- _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS
 - gettext.h, [96](#)
- __asm__
 - oom.c, [81](#)
- __sync_synchronize
 - oom.c, [81](#)
- _trace
 - assert.c, [32](#)
 - assert.h, [38](#)
- _trace_dynmsg
 - assert.c, [32](#)
 - assert.h, [38](#)
- _trace_msg
 - assert.c, [33](#)
 - assert.h, [39](#)
- ansi-color-codes.h, [17](#), [29](#)
 - BBLK, [18](#)
 - BBLU, [19](#)
 - BCYN, [19](#)
 - BGRN, [19](#)
 - BHBLK, [19](#)
 - BHBLU, [19](#)
 - BHCYN, [19](#)
 - BHGRN, [20](#)
 - BHMAG, [20](#)
 - BHRED, [20](#)
 - BHWHT, [20](#)
 - BHYEL, [20](#)
 - BLINK, [20](#)
 - BLK, [21](#)
 - BLKB, [21](#)
 - BLKHB, [21](#)
 - BLU, [21](#)
 - BLUB, [21](#)
 - BLUHB, [21](#)
 - BMAG, [22](#)
 - BOLD, [22](#)
 - BRED, [22](#)
 - BWHT, [22](#)
 - BYEL, [22](#)
 - CYN, [22](#)
 - CYNB, [23](#)
 - CYNHB, [23](#)
 - DIM, [23](#)
 - GRN, [23](#)
 - GRNB, [23](#)
 - GRNHB, [23](#)
 - HBLK, [24](#)
 - HBLU, [24](#)
 - HCYN, [24](#)
 - HGRN, [24](#)
 - HIDDEN, [24](#)
 - HMAG, [24](#)
 - HRED, [25](#)
 - HWHT, [25](#)
 - HYEL, [25](#)
 - MAG, [25](#)
 - MAGB, [25](#)
 - MAGHB, [25](#)
 - RED, [26](#)
 - REDB, [26](#)
 - REDHB, [26](#)
 - RESET, [26](#)
 - REVERSE, [26](#)
 - STRIKE, [26](#)
 - UBLK, [27](#)
 - UBLU, [27](#)
 - UCYN, [27](#)
 - UGRN, [27](#)
 - UMAG, [27](#)
 - UNDERLINE, [27](#)
 - URED, [28](#)
 - UWHT, [28](#)
 - UYEL, [28](#)
 - WHT, [28](#)
 - WHTB, [28](#)
 - WHTHB, [28](#)
 - YEL, [29](#)
 - YELB, [29](#)
 - YELHB, [29](#)
- asprintf
 - memory.h, [60](#)
- ASSERT
 - assert.h, [36](#)
- assert.c, [30](#), [33](#)
 - _trace, [32](#)
 - _trace_dynmsg, [32](#)
 - _trace_msg, [33](#)
- assert.h, [35](#), [39](#)
 - _trace, [38](#)

- [_trace_dynmsg](#), 38
- [_trace_msg](#), 39
- [ASSERT](#), 36
- [DBG_ITRACE](#), 37
- [DBG_MSG](#), 37
- [DBG_PRINTF](#), 37
- [DBG_TRACE](#), 37
- BBLK
 - [ansi-color-codes.h](#), 18
- BBLU
 - [ansi-color-codes.h](#), 19
- BCYN
 - [ansi-color-codes.h](#), 19
- BGRN
 - [ansi-color-codes.h](#), 19
- BHBLK
 - [ansi-color-codes.h](#), 19
- BHBLU
 - [ansi-color-codes.h](#), 19
- BHCYN
 - [ansi-color-codes.h](#), 19
- BHGRN
 - [ansi-color-codes.h](#), 20
- BHMAG
 - [ansi-color-codes.h](#), 20
- BHRED
 - [ansi-color-codes.h](#), 20
- BHWHT
 - [ansi-color-codes.h](#), 20
- BHYEL
 - [ansi-color-codes.h](#), 20
- [bind_textdomain_codeset](#)
 - [gettext.h](#), 97
- [bindtextdomain](#)
 - [gettext.h](#), 97
- BLINK
 - [ansi-color-codes.h](#), 20
- BLK
 - [ansi-color-codes.h](#), 21
- BLKB
 - [ansi-color-codes.h](#), 21
- BLKHB
 - [ansi-color-codes.h](#), 21
- BLU
 - [ansi-color-codes.h](#), 21
- BLUB
 - [ansi-color-codes.h](#), 21
- BLUHB
 - [ansi-color-codes.h](#), 21
- BMAG
 - [ansi-color-codes.h](#), 22
- BOLD
 - [ansi-color-codes.h](#), 22
- BRED
 - [ansi-color-codes.h](#), 22
- BWHT
 - [ansi-color-codes.h](#), 22
- BYEL
 - [ansi-color-codes.h](#), 22
- [ansi-color-codes.h](#), 22
- calloc
 - [memory.h](#), 61
- children
 - [service_s](#), 14
- CompilDate
 - MemBlock, 8
- CompilTime
 - MemBlock, 8
- CYN
 - [ansi-color-codes.h](#), 22
- CYNB
 - [ansi-color-codes.h](#), 23
- CYNHB
 - [ansi-color-codes.h](#), 23
- [dbg_asprintf](#)
 - [memdbg.c](#), 42
 - [memdbg.h](#), 52
- [dbg_calloc](#)
 - [memdbg.c](#), 43
 - [memdbg.h](#), 53
- [dbg_free](#)
 - [memdbg.c](#), 44
 - [memdbg.h](#), 54
- DBG_ITRACE
 - [assert.h](#), 37
- [dbg_malloc](#)
 - [memdbg.c](#), 45
 - [memdbg.h](#), 55
- DBG_MSG
 - [assert.h](#), 37
- DBG_PRINTF
 - [assert.h](#), 37
- [dbg_realloc](#)
 - [memdbg.c](#), 46
 - [memdbg.h](#), 56
- [dbg_strdup](#)
 - [memdbg.c](#), 47
 - [memdbg.h](#), 57
- DBG_TRACE
 - [assert.h](#), 37
- dcgettext
 - [gettext.h](#), 97
- dcngettext
 - [gettext.h](#), 97
- dcnpgettext
 - [gettext.h](#), 97
- dcppgettext
 - [gettext.h](#), 98
- dgettext
 - [gettext.h](#), 98
- DIM
 - [ansi-color-codes.h](#), 23
- dngettext
 - [gettext.h](#), 98
- dnppgettext
 - [gettext.h](#), 98

- dnpgettext_expr
 - gettext.h, 99
- dpgettext
 - gettext.h, 99
- dpgettext_expr
 - gettext.h, 99
- File
 - MemBlock, 8
- free
 - memory.h, 61
- Function
 - MemBlock, 8
- function
 - service_s, 14
- gettext
 - gettext.h, 99
- gettext.h, 95, 102
 - _LIBGETTEXT_HAVE_VARIABLE_SIZE_ARRAYS, 96
 - bind_textdomain_codeset, 97
 - bindtextdomain, 97
 - dcgettext, 97
 - dcngettext, 97
 - dcnpgettext, 97
 - dcpgettext, 98
 - dgettext, 98
 - dngettext, 98
 - dnpgettext, 98
 - dnpgettext_expr, 99
 - dpgettext, 99
 - dpgettext_expr, 99
 - gettext, 99
 - GETTEXT_CONTEXT_GLUE, 100
 - gettext_noop, 100
 - ngettext, 100
 - npgettext, 100
 - npgettext_expr, 100
 - pgettext, 101
 - pgettext_expr, 101
 - textdomain, 101
- GETTEXT_CONTEXT_GLUE
 - gettext.h, 100
- gettext_noop
 - gettext.h, 100
- GRN
 - ansi-color-codes.h, 23
- GRNB
 - ansi-color-codes.h, 23
- GRNHB
 - ansi-color-codes.h, 23
- handle
 - modules_s, 13
- HBLK
 - ansi-color-codes.h, 24
- HBLU
 - ansi-color-codes.h, 24
- HCYN
 - ansi-color-codes.h, 24
- HELP_DESC
 - mkkernel-opt.c, 107
- HELP_name
 - mkkernel-opt.c, 107
- HGRN
 - ansi-color-codes.h, 24
- HIDDEN
 - ansi-color-codes.h, 24
- HMAG
 - ansi-color-codes.h, 24
- HRED
 - ansi-color-codes.h, 25
- HWHT
 - ansi-color-codes.h, 25
- HYEL
 - ansi-color-codes.h, 25
- Line
 - MemBlock, 8
- LOAD_OPTS_DESC
 - mkkernel-opt.c, 107
- LOAD_OPTS_NAME
 - mkkernel-opt.c, 107
- LOAD_OPTS_name
 - mkkernel-opt.c, 108
- LOAD_OPTS_pfx
 - mkkernel-opt.c, 108
- MAG
 - ansi-color-codes.h, 25
- MAGB
 - ansi-color-codes.h, 25
- MAGHB
 - ansi-color-codes.h, 25
- main
 - mkkernel.c, 127
- malloc
 - memory.h, 61
- MemBlock, 7
 - CompileDate, 8
 - CompileTime, 8
 - File, 8
 - Function, 8
 - Line, 8
 - Next, 9
 - Prev, 9
 - Ptr, 9
 - Size, 9
- memdbg.c, 41, 48
 - dbg_asprintf, 42
 - dbg_calloc, 43
 - dbg_free, 44
 - dbg_malloc, 45
 - dbg_realloc, 46
 - dbg_strdup, 47
- memdbg.h, 50, 58
 - dbg_asprintf, 52

- dbg_calloc, 53
- dbg_free, 54
- dbg_malloc, 55
- dbg_realloc, 56
- dbg_strdup, 57
- memory.h, 59, 62
 - asprintf, 60
 - calloc, 61
 - free, 61
 - malloc, 61
 - memreport, 61
 - realloc, 62
 - strdup, 62
- memreport
 - memory.h, 61
- memtrack.c, 63, 68
 - memtrack_addblock, 64
 - memtrack_delblock, 65
 - memtrack_dumpblocks, 66
 - memtrack_getallocatedblocks, 67
 - memtrack_getallocatedRAM, 67
 - memtrack_getblocksize, 67
 - memtrack_reset, 64
- memtrack.h, 74, 79
 - memtrack_addblock, 76
 - memtrack_delblock, 77
 - memtrack_dumpblocks, 77
 - memtrack_getallocatedblocks, 78
 - memtrack_getallocatedRAM, 78
 - memtrack_getblocksize, 78
 - TMemBlock, 75
- memtrack_addblock
 - memtrack.c, 64
 - memtrack.h, 76
- memtrack_delblock
 - memtrack.c, 65
 - memtrack.h, 77
- memtrack_dumpblocks
 - memtrack.c, 66
 - memtrack.h, 77
- memtrack_getallocatedblocks
 - memtrack.c, 67
 - memtrack.h, 78
- memtrack_getallocatedRAM
 - memtrack.c, 67
 - memtrack.h, 78
- memtrack_getblocksize
 - memtrack.c, 67
 - memtrack.h, 78
- memtrack_reset
 - memtrack.c, 64
- mkkernel-opt.c, 105, 116
 - HELP_DESC, 107
 - HELP_name, 107
 - LOAD_OPTS_DESC, 107
 - LOAD_OPTS_NAME, 107
 - LOAD_OPTS_name, 108
 - LOAD_OPTS_pfx, 108
- mkkernel_full_usage, 108
- mkkernel_packager_info, 108
- mkkernel_short_usage, 108
- mkkernelOptions, 113
- MODULE_PATH_DESC, 108
- MODULE_PATH_FLAGS, 109
- MODULE_PATH_NAME, 109
- MODULE_PATH_name, 109
- MORE_HELP_DESC, 109
- MORE_HELP_FLAGS, 109
- MORE_HELP_name, 110
- NO_LOAD_OPTS_name, 110
- NULL, 110
- OPTION_CODE_COMPILE, 110
- option_usage_fp, 114
- optionBooleanVal, 114
- optionNestedVal, 114
- optionNumericVal, 114
- optionPagedUsage, 114
- optionPrintVersion, 114
- optionResetOpt, 115
- optionStackArg, 115
- optionTimeDate, 115
- optionTimeVal, 115
- optionUnstackArg, 115
- optionVendorOption, 115
- OPTPROC_BASE, 110
- PKGDATA DIR, 110
- SAVE_OPTS_DESC, 111
- SAVE_OPTS_name, 111
- translate_option_strings, 111
- VER_DESC, 111
- VER_FLAGS, 111
- VER_name, 111
- VER_PROC, 112
- zBugsAddr, 112
- zCopyright, 112
- zDetail, 112
- zExplain, 112
- zFullVersion, 112
- zLicenseDescrip, 113
- zPROGNAME, 113
- zRcName, 113
- zUsageTitle, 113
- mkkernel.c, 125, 128
 - _, 126
 - main, 127
 - MODULE_PATH_DEFAULT, 126
 - MODULE_PATH_ENV, 126
 - PATH_MAX, 127
- mkkernel_full_usage
 - mkkernel-opt.c, 108
- mkkernel_packager_info
 - mkkernel-opt.c, 108
- mkkernel_short_usage
 - mkkernel-opt.c, 108
- mkkernelOptions
 - mkkernel-opt.c, 113

- mkmod.h, [130](#), [131](#)
 - mkmod_api_t, [131](#)
- mkmod_api_s, [10](#)
 - mkmod_function, [10](#)
- mkmod_api_t
 - mkmod.h, [131](#)
- mkmod_function
 - mkmod_api_s, [10](#)
- mkmodgtk.c, [131](#), [134](#)
 - _, [133](#)
 - module_api, [133](#)
 - onLoad, [133](#)
 - onUnload, [133](#)
- mkmodtty.c, [135](#), [137](#)
 - _, [136](#)
 - module_api, [137](#)
 - onLoad, [136](#)
 - onUnload, [136](#)
- modinfo
 - modules_s, [13](#)
- modmgr.c, [138](#), [143](#)
 - _, [139](#)
 - modmgr_addpath, [140](#)
 - modmgr_getpath, [140](#)
 - modmgr_getsymbol, [141](#)
 - modmgr_insertpath, [141](#)
 - modmgr_list, [141](#)
 - modmgr_load, [141](#)
 - modmgr_setpath, [142](#)
 - modmgr_unload, [142](#)
 - modules_t, [140](#)
 - PATH_MAX, [139](#)
- modmgr.h, [148](#), [153](#)
 - modmgr_addpath, [150](#)
 - MODMGR_GETFUNCTION, [149](#)
 - modmgr_getpath, [150](#)
 - modmgr_getsymbol, [151](#)
 - modmgr_insertpath, [151](#)
 - modmgr_list, [151](#)
 - MODMGR_LOAD, [149](#)
 - modmgr_load, [151](#)
 - modmgr_module_t, [150](#)
 - modmgr_setpath, [152](#)
 - modmgr_unload, [152](#)
- modmgr_addpath
 - modmgr.c, [140](#)
 - modmgr.h, [150](#)
- MODMGR_GETFUNCTION
 - modmgr.h, [149](#)
- modmgr_getpath
 - modmgr.c, [140](#)
 - modmgr.h, [150](#)
- modmgr_getsymbol
 - modmgr.c, [141](#)
 - modmgr.h, [151](#)
- modmgr_insertpath
 - modmgr.c, [141](#)
 - modmgr.h, [151](#)
- modmgr_list
 - modmgr.c, [141](#)
 - modmgr.h, [151](#)
- MODMGR_LOAD
 - modmgr.h, [149](#)
- modmgr_load
 - modmgr.c, [141](#)
 - modmgr.h, [151](#)
- modmgr_module_t
 - modmgr.h, [150](#)
- modmgr_setpath
 - modmgr.c, [142](#)
 - modmgr.h, [152](#)
- modmgr_unload
 - modmgr.c, [142](#)
 - modmgr.h, [152](#)
- module.h, [153](#), [155](#)
 - moduleinfo_t, [154](#)
- module_api
 - mkmodgtk.c, [133](#)
 - mkmodtty.c, [137](#)
- MODULE_PATH_DEFAULT
 - kernel.c, [126](#)
- MODULE_PATH_DESC
 - kernel-opt.c, [108](#)
- MODULE_PATH_ENV
 - kernel.c, [126](#)
- MODULE_PATH_FLAGS
 - kernel-opt.c, [109](#)
- MODULE_PATH_NAME
 - kernel-opt.c, [109](#)
- MODULE_PATH_name
 - kernel-opt.c, [109](#)
- moduleAuthor
 - moduleinfo_s, [11](#)
- moduleDesc
 - moduleinfo_s, [11](#)
- moduleEmail
 - moduleinfo_s, [11](#)
- moduleinfo_s, [10](#)
 - moduleAuthor, [11](#)
 - moduleDesc, [11](#)
 - moduleEmail, [11](#)
 - moduleLicense, [11](#)
 - moduleMajor, [11](#)
 - moduleMinor, [11](#)
 - moduleName, [12](#)
 - modulePatch, [12](#)
 - moduleURL, [12](#)
- moduleinfo_t
 - module.h, [154](#)
- moduleLicense
 - moduleinfo_s, [11](#)
- moduleMajor
 - moduleinfo_s, [11](#)
- moduleMinor
 - moduleinfo_s, [11](#)
- moduleName

- moduleinfo_s, 12
- modulePatch
 - moduleinfo_s, 12
- modules_s, 12
 - handle, 13
 - modinfo, 13
 - next, 13
- modules_t
 - modmgr.c, 140
- moduleURL
 - moduleinfo_s, 12
- MORE_HELP_DESC
 - mkkernel-opt.c, 109
- MORE_HELP_FLAGS
 - mkkernel-opt.c, 109
- MORE_HELP_name
 - mkkernel-opt.c, 110
- name
 - service_s, 14
- nbArgs
 - service_s, 15
- Next
 - MemBlock, 9
- next
 - modules_s, 13
 - service_s, 15
- ngettext
 - gettext.h, 100
- NO_LOAD_OPTS_name
 - mkkernel-opt.c, 110
- npgettext
 - gettext.h, 100
- npgettext_expr
 - gettext.h, 100
- NULL
 - mkkernel-opt.c, 110
- onLoad
 - mkmodgtk.c, 133
 - mkmodtty.c, 136
- onUnload
 - mkmodgtk.c, 133
 - mkmodtty.c, 136
- oom.c, 80, 85
 - _GNU_SOURCE, 81
 - __asm__, 81
 - __sync_synchronize, 81
 - oomtest_config, 82
 - oomtest_disable, 83
 - oomtest_enable, 83
 - oomtest_enabled, 82
 - oomtest_fill, 84
 - oomtest_free, 85
 - RAMBLOCKS_MAX, 81
- oom.h, 89, 94
 - oomtest_config, 91
 - oomtest_disable, 92
 - oomtest_enable, 92
 - oomtest_enabled, 92
 - oomtest_fill, 93
 - oomtest_free, 94
 - RAMLIMIT_HARD, 90
 - RAMLIMIT_SOFT, 91
- oomtest_config
 - oom.c, 82
 - oom.h, 91
- oomtest_disable
 - oom.c, 83
 - oom.h, 92
- oomtest_enable
 - oom.c, 83
 - oom.h, 92
- oomtest_enabled
 - oom.c, 82
 - oom.h, 92
- oomtest_fill
 - oom.c, 84
 - oom.h, 93
- oomtest_free
 - oom.c, 85
 - oom.h, 94
- OPTION_CODE_COMPILE
 - mkkernel-opt.c, 110
- option_usage_fp
 - mkkernel-opt.c, 114
- optionBooleanVal
 - mkkernel-opt.c, 114
- optionNestedVal
 - mkkernel-opt.c, 114
- optionNumericVal
 - mkkernel-opt.c, 114
- optionPagedUsage
 - mkkernel-opt.c, 114
- optionPrintVersion
 - mkkernel-opt.c, 114
- optionResetOpt
 - mkkernel-opt.c, 115
- optionStackArg
 - mkkernel-opt.c, 115
- optionTimeDate
 - mkkernel-opt.c, 115
- optionTimeVal
 - mkkernel-opt.c, 115
- optionUnstackArg
 - mkkernel-opt.c, 115
- optionVendorOption
 - mkkernel-opt.c, 115
- OPTPROC_BASE
 - mkkernel-opt.c, 110
- PATH_MAX
 - mkkernel.c, 127
 - modmgr.c, 139
- pgettext
 - gettext.h, 101
- pgettext_expr
 - gettext.h, 101

- PKGDATA DIR
 - mkkernel-opt.c, [110](#)
- Prev
 - MemBlock, [9](#)
- Ptr
 - MemBlock, [9](#)
- RAMBLOCKS_MAX
 - oom.c, [81](#)
- RAMLIMIT_HARD
 - oom.h, [90](#)
- RAMLIMIT_SOFT
 - oom.h, [91](#)
- realloc
 - memory.h, [62](#)
- RED
 - ansi-color-codes.h, [26](#)
- REDB
 - ansi-color-codes.h, [26](#)
- REDHB
 - ansi-color-codes.h, [26](#)
- RESET
 - ansi-color-codes.h, [26](#)
- REVERSE
 - ansi-color-codes.h, [26](#)
- REVISION
 - revision.h, [156](#)
- revision.h, [155](#), [156](#)
 - REVISION, [156](#)
- SAVE_OPTS_DESC
 - mkkernel-opt.c, [111](#)
- SAVE_OPTS_name
 - mkkernel-opt.c, [111](#)
- service_s, [14](#)
 - children, [14](#)
 - function, [14](#)
 - name, [14](#)
 - nbArgs, [15](#)
 - next, [15](#)
- service_t
 - svcmgr.c, [158](#)
- Size
 - MemBlock, [9](#)
- strdup
 - memory.h, [62](#)
- STRIKE
 - ansi-color-codes.h, [26](#)
- svcfunc_t
 - svcmgr.h, [163](#)
- svcmgr.c, [156](#), [159](#)
 - _, [157](#)
 - service_t, [158](#)
 - svcmgr_call, [158](#)
 - svcmgr_dump, [158](#)
 - svcmgr_register, [158](#)
 - svcmgr_unregister, [159](#)
- svcmgr.h, [162](#), [164](#)
 - svcfunc_t, [163](#)
 - svcmgr_call, [163](#)
 - svcmgr_register, [163](#)
 - svcmgr_unregister, [164](#)
- svcmgr_call
 - svcmgr.c, [158](#)
 - svcmgr.h, [163](#)
- svcmgr_dump
 - svcmgr.c, [158](#)
- svcmgr_register
 - svcmgr.c, [158](#)
 - svcmgr.h, [163](#)
- svcmgr_unregister
 - svcmgr.c, [159](#)
 - svcmgr.h, [164](#)
- textdomain
 - gettext.h, [101](#)
- TMemBlock
 - memtrack.h, [75](#)
- translate_option_strings
 - mkkernel-opt.c, [111](#)
- UBLK
 - ansi-color-codes.h, [27](#)
- UBLU
 - ansi-color-codes.h, [27](#)
- UCYN
 - ansi-color-codes.h, [27](#)
- UGRN
 - ansi-color-codes.h, [27](#)
- UMAG
 - ansi-color-codes.h, [27](#)
- UNDERLINE
 - ansi-color-codes.h, [27](#)
- URED
 - ansi-color-codes.h, [28](#)
- UWHT
 - ansi-color-codes.h, [28](#)
- UYEL
 - ansi-color-codes.h, [28](#)
- VER_DESC
 - mkkernel-opt.c, [111](#)
- VER_FLAGS
 - mkkernel-opt.c, [111](#)
- VER_name
 - mkkernel-opt.c, [111](#)
- VER_PROC
 - mkkernel-opt.c, [112](#)
- WHT
 - ansi-color-codes.h, [28](#)
- WHTB
 - ansi-color-codes.h, [28](#)
- WHTHB
 - ansi-color-codes.h, [28](#)
- YEL
 - ansi-color-codes.h, [29](#)

YELB

ansi-color-codes.h, [29](#)

YELHB

ansi-color-codes.h, [29](#)

zBugsAddr

mkernel-opt.c, [112](#)

zCopyright

mkernel-opt.c, [112](#)

zDetail

mkernel-opt.c, [112](#)

zExplain

mkernel-opt.c, [112](#)

zFullVersion

mkernel-opt.c, [112](#)

zLicenseDescrip

mkernel-opt.c, [113](#)

zPROGNAME

mkernel-opt.c, [113](#)

zRcName

mkernel-opt.c, [113](#)

zUsageTitle

mkernel-opt.c, [113](#)