mpatrol reference card

The mpatrol library can read certain options at run-time from an environment variable called `MPATROL_OPTIONS`. This variable must contain one or more valid option keywords from the list below and must be no longer than 1024 characters in length. If `MPATROL_OPTIONS` is unset or empty then the default settings will be used.
The `LOGDIR`, `PROFDIR` and `TRACEDIR` environment variables are also read in order to determine where the log file, profiling output file and tracing output file should go. Note that if they are set then the default filenames for the log file, profiling output file and tracing output file will also be changed.

Library behaviour

`HELP` Displays a quick-reference option summary.

`PROGFILE=<`***string***`>` Specifies an alternative filename with which to locate the executable file containing the program's symbols.

`CHECK=<`***unsigned range***`>` Specifies a range of allocation indices at which to check the integrity of free memory and overflow buffers.

`EDIT` Specifies that a text editor should be invoked to edit any relevant source files that are associated with any warnings or errors when they occur.

`LIST` Specifies that a context listing should be shown for any relevant source files that are associated with any warnings or errors when they occur.

`DEFALIGN=<`***unsigned integer***`>` Specifies the default alignment for general-purpose memory allocations, which must be a power of two.

`NOPROTECT` Specifies that the mpatrol library's internal data structures should not be made read-only after every memory allocation, reallocation or deallocation.

`SAFESIGNALS` Instructs the library to save and replace certain signal handlers during the execution of library code and to restore them afterwards.

`CHECKFORK` Checks at every call to see if the process has been forked in case new log, profiling and tracing output files need to be started.

`USEMMAP` Specifies that the library should use `mmap()` instead of `sbrk()` to allocate user memory on UNIX platforms.

Logging and tracing

`LOGFILE=<`***string***`>` Specifies an alternative file in which to place all diagnostics from the mpatrol library.

`LOGALLOCS` Specifies that all memory allocations are to be logged and sent to the log file.

`LOGREALLOCS` Specifies that all memory reallocations are to be logged and sent to the log file.

`LOGFREES` Specifies that all memory deallocations are to be logged and sent to the log file.

`LOGMEMORY` Specifies that all memory operations are to be logged and sent to the log file.

`LOGALL` Equivalent to the `LOGALLOCS`, `LOGREALLOCS`, `LOGFREES` and `LOGMEMORY` options specified together.

`TRACE` Specifies that all memory allocations are to be traced and sent to the tracing output file.

`TRACEFILE=<`***string***`>` Specifies an alternative file in which to place all memory allocation tracing information from the mpatrol library.

`LEAKTABLE` Specifies that the leak table should be automatically used and a leak table summary should be displayed at the end of program execution.

`SHOWFREE` Specifies that a summary of all of the free memory blocks should be displayed at the end of program execution.

`SHOWFREED` Specifies that a summary of all of the freed memory allocations should be displayed at the end of program execution.

`SHOWUNFREED` Specifies that a summary of all of the unfreed memory allocations should be displayed at the end of program execution.

`SHOWMAP` Specifies that a memory map of the entire heap should be displayed at the end of program execution.

`SHOWSYMBOLS` Specifies that a summary of all of the function symbols read from the program's executable file should be displayed at the end of program execution.

`SHOWALL` Equivalent to the `SHOWFREE`, `SHOWFREED`, `SHOWUNFREED`, `SHOWMAP` and `SHOWSYMBOLS` options specified together.

`USEDEBUG` Specifies that any debugging information in the executable file should be used to obtain additional source-level information.

General errors

`CHECKALLOCS` Checks that no attempt is made to allocate a block of memory of size zero.

`CHECKREALLOCS` Checks that no attempt is made to reallocate a `NULL` pointer or resize an existing block of memory to size zero.

`CHECKFREES` Checks that no attempt is made to deallocate a `NULL` pointer.

`CHECKMEMORY` Checks that no attempt is made to perform a zero-length memory operation on a `NULL` pointer.

`CHECKALL` Equivalent to the `CHECKALLOCS`, `CHECKREALLOCS`, `CHECKFREES` and `CHECKMEMORY` options specified together.

`ALLOCBYTE=<`***unsigned integer***`>` Specifies an 8-bit byte pattern with which to prefill newly-allocated memory.

`FREEBYTE=<`***unsigned integer***`>` Specifies an 8-bit byte pattern with which to prefill newly-freed memory.

`NOFREE=<`***unsigned integer***`>` Specifies that a number of recently-freed memory allocations should be prevented from being returned to the free memory pool.

`PRESERVE` Specifies that any reallocated or freed memory allocations should preserve their original contents.

Overwrites and underwrites

`OFLOWSIZE=<`***unsigned integer***`>` Specifies the size in bytes to use for all overflow buffers, which must be a power of two.

`OFLOWBYTE=<`***unsigned integer***`>` Specifies an 8-bit byte pattern with which to fill the overflow buffers of all memory allocations.

`OFLOWWATCH` Specifies that watch point areas should be used for overflow buffers rather than filling with the overflow byte.

`PAGEALLOC=<LOWER|UPPER>` Specifies that each individual memory allocation should occupy at least one page of virtual memory and should be placed at the lowest or highest point within these pages.

`ALLOWOFLOW` Specifies that a warning rather than an error should be produced if any memory operation function overflows the boundaries of a memory allocation, and that the operation should still be performed.

Using with a debugger

`ALLOCSTOP=<`***unsigned integer***`>` Specifies an allocation index at which to stop the program when it is being allocated.

`REALLOCSTOP=<`***unsigned integer***`>` Specifies an allocation index at which to stop the program when a memory allocation is being reallocated.

`FREESTOP=<`***unsigned integer***`>` Specifies an allocation index at which to stop the program when it is being freed.

Testing

`LIMIT=<`***unsigned integer***`>` Specifies the limit in bytes at which all memory allocations should fail if the total allocated memory should increase beyond this.

`FAILFREQ=<`***unsigned integer***`>` Specifies the frequency at which all memory allocations will randomly fail.

`FAILSEED=<`***unsigned integer***`>` Specifies the random number seed which will be used when determining which memory allocations will randomly fail.

`UNFREEDABORT=<`***unsigned integer***`>` Specifies the minimum number of unfreed allocations at which to abort the program just before program termination.

Profiling

`PROF` Specifies that all memory allocations are to be profiled and sent to the profiling output file.

`PROFFILE=<`***string***`>` Specifies an alternative file in which to place all memory allocation profiling information from the mpatrol library.

`AUTOSAVE=<`***unsigned integer***`>` Specifies the frequency at which to periodically write the profiling data to the profiling output file.

`SMALLBOUND=<`***unsigned integer***`>` Specifies the limit in bytes up to which memory allocations should be classified as small allocations for profiling purposes.

`MEDIUMBOUND=<`***unsigned integer***`>` Specifies the limit in bytes up to which memory allocations should be classified as medium allocations for profiling purposes.

`LARGEBOUND=<`***unsigned integer***`>` Specifies the limit in bytes up to which memory allocations should be classified as large allocations for profiling purposes.

All of the function definitions in `mpatrol.h` can be disabled by defining the `NDEBUG` preprocessor macro, which is the same macro used to control the behaviour of the `assert()` function. If `NDEBUG` is defined then no macro redefinition of functions will take place and all special mpatrol library functions will evaluate to empty statements. The `mpalloc.h` header file will also be included in this case. It is intended that the `NDEBUG` preprocessor macro be defined in release builds.

The `MP_MALLOC()` family of functions that are defined in `mpalloc.h` are also defined in `mpatrol.h` when `NDEBUG` is not defined. The mpatrol versions of these functions contain more debugging information than the mpalloc versions do, but they do not call the allocation failure handler when no more memory is available (they cause the `OUTMEM` error message to be given instead).

There may be problems during preprocessing when the preprocessor macros defining the replacement C++ operators in `mpatrol.h` are used. If this is the case then either the `MP_NOCPLUSPLUS` preprocessor macro can be defined to disable all C++ support, or the `MP_NONEWDELETE` preprocessor macro can be defined to prevent the debugging versions of `operator new` and `operator delete` from being used by default. The preprocessor macros `MP_NEW`, `MP_NEW_NOTHROW` and `MP_DELETE` will then have to be used instead.

On systems that support it, global functions (with C linkage) in an executable file or shared library whose names begin with `__mp_init_` will be noted when the mpatrol library first starts up and is reading the symbols. Such functions will then be called as soon as the mpatrol library is initialised, which can be useful if the initialisation occurs before `main()` is called. These functions must accept no arguments and must return no value. Similar behaviour exists for global functions whose names begin with `__mp_fini_`, except that such functions will be executed when the mpatrol library terminates.

C dynamic memory allocation functions

`malloc()`	Allocates memory.
`calloc()`	Allocates zero-filled memory.
`memalign()`	Allocates memory with a specified alignment.
`valloc()`	Allocates page-aligned memory.
`pvalloc()`	Allocates a number of pages.
`alloca()`	Allocates temporary memory.
`strdup()`	Duplicates a string.
`strndup()`	Duplicates a string with a maximum length.
`strsave()`	Duplicates a string.
`strnsave()`	Duplicates a string with a maximum length.
`strdupa()`	Duplicates a string.
`strndupa()`	Duplicates a string with a maximum length.
`realloc()`	Resizes memory.
`reallocf()`	Resizes memory and frees on failure.
`recalloc()`	Resizes memory allocated by `calloc()`.
`expand()`	Resizes memory but does not relocate it.
`free()`	Frees memory.
`cfree()`	Frees memory allocated by `calloc()`.
`dealloca()`	Explicitly frees temporary memory.

C dynamic memory extension functions

`xmalloc()`	Allocates memory without failure.
`xcalloc()`	Allocates zero-filled memory without failure.
`xstrdup()`	Duplicates a string without failure.
`xrealloc()`	Resizes memory without failure.
`xfree()`	Frees memory.

C dynamic memory alternative functions

`MP_MALLOC()`	Allocates memory without failure.
`MP_CALLOC()`	Allocates zero-filled memory without failure.
`MP_STRDUP()`	Duplicates a string without failure.
`MP_REALLOC()`	Resizes memory without failure.
`MP_FREE()`	Frees memory.
`MP_FAILURE()`	Sets the allocation failure handler.

C++ dynamic memory allocation functions

`operator new`	Allocates memory.
`operator new[]`	Allocates memory for an array.
`operator delete`	Frees memory.
`operator delete[]`	Frees memory allocated by `new[]`.
`set_new_handler()`	Sets up an allocation failure handler.

C memory operation functions

`memset()`	Fills memory with a specific byte.
`bzero()`	Fills memory with the zero byte.
`memccpy()`	Copies memory up to a specific byte.
`memcpy()`	Copies non-overlapping memory.
`memmove()`	Copies possibly-overlapping memory.
`bcopy()`	Copies possibly-overlapping memory.
`memcmp()`	Compares two blocks of memory.
`bcmp()`	Compares two blocks of memory.
`memchr()`	Searches memory for a specific byte.
`memmem()`	Searches memory for specific bytes.

mpatrol library functions

`__mp_atexit()`	Registers termination functions.
`__mp_setoption()`	Sets an mpatrol library option.
`__mp_getoption()`	Returns an mpatrol library option.
`__mp_libversion()`	Returns the mpatrol library version.
`__mp_strerror()`	Returns an error message string.
`__mp_function()`	Returns an allocation type function name.
`__mp_setuser()`	Sets the user data for an allocation.
`__mp_setmark()`	Sets the marked flag for an allocation.
`__mp_info()`	Returns information for an allocation.
`__mp_syminfo()`	Returns symbol information for an address.
`__mp_symbol()`	Returns symbol name for an address.
`__mp_printinfo()`	Displays information for an allocation.
`__mp_snapshot()`	Returns the current heap event number.
`__mp_iterate()`	Iterates over allocations in the heap.
`__mp_iterateall()`	Iterates over all allocations in the heap.
`__mp_addallocentry()`	Adds an allocation to the leak table.
`__mp_addfreeentry()`	Adds a deallocation to the leak table.

`__mp_clearleaktable()`	Clears the leak table.
`__mp_startleaktable()`	Starts automatic leak table entries.
`__mp_stopleaktable()`	Stops automatic leak table entries.
`__mp_leaktable()`	Displays the leak table.
`__mp_memorymap()`	Displays a map of memory in the heap.
`__mp_summary()`	Displays a summary of library statistics.
`__mp_stats()`	Returns statistics about the heap.
`__mp_check()`	Validates memory in the heap.
`__mp_prologue()`	Sets up an allocation prologue handler.
`__mp_epilogue()`	Sets up an allocation epilogue handler.
`__mp_nomemory()`	Sets up an allocation failure handler.
`__mp_printf()`	Writes user data to the log file.
`__mp_vprintf()`	Writes user data to the log file.
`__mp_locprintf()`	Logs user data and the location.
`__mp_vlocprintf()`	Logs user data and the location.
`__mp_logmemory()`	Displays a hex dump of memory.
`__mp_logstack()`	Displays the current call stack.
`__mp_logaddr()`	Displays information for an allocation.
`__mp_edit()`	Invokes a text editor on a source file.
`__mp_list()`	Lists a source file at a specific line.
`__mp_view()`	Edits or lists a source file.
`__mp_readcontents()`	Reads the contents of a memory allocation.
`__mp_writecontents()`	Writes the contents of a memory allocation.
`__mp_cmpcontents()`	Compares the contents of a memory allocation.
`__mp_remcontents()`	Removes the contents of a memory allocation.

mpatrol library variables

`__mp_errno`	Contains the most recent error code.

The following table lists the warnings and errors that are likely to appear in the mpatrol log file when problems with dynamic memory allocations and memory operations occur. Other types of warnings and errors may also appear in the log file, but they are likely to be associated with parsing options and reading symbols from executable files and so should be self-explanatory.

Error abbreviation codes

`ALLOVF`	Allocation has a corrupted overflow buffer.
`ALLZER`	Attempt to create an allocation of size 0.
`BADALN`	Alignment is not a power of two.
`FRDCOR`	Freed allocation has memory corruption.
`FRDOPN`	Attempt to perform operation on freed memory.
`FRDOVF`	Freed allocation has a corrupted overflow buffer.
`FRECOR`	Free memory corruption.
`FREMRK`	Attempt to free a marked memory allocation.
`FRENUL`	Attempt to free a `NULL` pointer.
`FREOPN`	Attempt to perform operation on free memory.
`ILLMEM`	Illegal memory access.
`INCOMP`	Attempt to resize or free memory allocated with an incompatible function.
`MAXALN`	Alignment is greater than the system page size.
`MISMAT`	Attempt to resize or free memory not pointing to the start of a memory allocation.
`NOTALL`	Pointer has not been allocated.
`NULOPN`	Attempt to perform operation on a `NULL` pointer.
`OUTMEM`	Out of memory.
`PRVFRD`	Attempt to resize of free memory that has previously been freed.
`RNGOVF`	Attempt to perform a memory operation that overflows a memory allocation.
`RNGOVL`	Attempt to perform a non-overlapping memory operation that overlaps.
`RSZNUL`	Attempt to resize a `NULL` pointer.
`RSZZER`	Attempt to resize an allocation to size 0.
`STROVF`	Attempt to perform a string operation that overflows a memory allocation.
`ZERALN`	Alignment 0 is invalid.

The commands that are distributed with the mpatrol library all parse their command line options in a similar way to the UNIX `getopt()` function. Only the long names of the options are shown here. If an option has a single character equivalent it will be listed in the —**help** output. Options that accept numeric arguments can have their value specified in binary, octal, decimal or hexadecimal notation.

mpatrol command options

—**alloc-byte** <*unsigned integer*>	See `ALLOCBYTE`.
—**alloc-stop** <*unsigned integer*>	See `ALLOCSTOP`.
—**allow-oflow**	See `ALLOWOFLOW`.
—**auto-save** <*unsigned integer*>	See `AUTOSAVE`.
—**check** <*unsigned range*>	See `CHECK`.
—**check-all**	See `CHECKALL`.
—**check-allocs**	See `CHECKALLOCS`.
—**check-fork**	See `CHECKFORK`.
—**check-frees**	See `CHECKFREES`.
—**check-memory**	See `CHECKMEMORY`.
—**check-reallocs**	See `CHECKREALLOCS`.
—**def-align** <*unsigned integer*>	See `DEFALIGN`.
—**dynamic** Specifies that programs which were not linked with the mpatrol library should also be traced, but only if they were dynamically linked.	
—**edit**	See `EDIT`.
—**fail-freq** <*unsigned integer*>	See `FAILFREQ`.
—**fail-seed** <*unsigned integer*>	See `FAILSEED`.
—**free-byte** <*unsigned integer*>	See `FREEBYTE`.
—**free-stop** <*unsigned integer*>	See `FREESTOP`.
—**help** Displays a quick-reference option summary.	
—**large-bound** <*unsigned integer*>	See `LARGEBOUND`.
—**leak-table**	See `LEAKTABLE`.
—**limit** <*unsigned integer*>	See `LIMIT`.
—**list**	See `LIST`.
—**log-all**	See `LOGALL`.
—**log-allocs**	See `LOGALLOCS`.
—**log-file** <*string*>	See `LOGFILE`.
—**log-frees**	See `LOGFREES`.
—**log-memory**	See `LOGMEMORY`.
—**log-reallocs**	See `LOGREALLOCS`.
—**medium-bound** <*unsigned integer*>	See `MEDIUMBOUND`.
—**no-free** <*unsigned integer*>	See `NOFREE`.
—**no-protect**	See `NOPROTECT`.
—**oflow-byte** <*unsigned integer*>	See `OFLOWBYTE`.
—**oflow-size** <*unsigned integer*>	See `OFLOWSIZE`.
—**oflow-watch**	See `OFLOWWATCH`.
—**page-alloc-lower**	See `PAGEALLOC=LOWER`.
—**page-alloc-upper**	See `PAGEALLOC=UPPER`.
—**preserve**	See `PRESERVE`.
—**prof**	See `PROF`.
—**prof-file** <*string*>	See `PROFFILE`.
—**prog-file** <*string*>	See `PROGFILE`.
—**read-env** Reads and passes through the contents of the `MPATROL_OPTIONS` environment variable.	

—**realloc-stop** <*unsigned integer*> See `REALLOCSTOP`.

—**safe-signals** See `SAFESIGNALS`.

—**show-all** See `SHOWALL`.

—**show-env** Displays the contents of the `MPATROL_OPTIONS` environment variable.

—**show-free** See `SHOWFREE`.

—**show-freed** See `SHOWFREED`.

—**show-map** See `SHOWMAP`.

—**show-symbols** See `SHOWSYMBOLS`.

—**show-unfreed** See `SHOWUNFREED`.

—**small-bound** <*unsigned integer*> See `SMALLBOUND`.

—**threads** Specifies that the program to be run is multi-threaded if the —**dynamic** option is used.

—**trace** See `TRACE`.

—**trace-file** <*string*> See `TRACEFILE`.

—**unfreed-abort** <*unsigned integer*> See `UNFREEDABORT`.

—**use-debug** See `USEDEBUG`.

—**use-mmap** See `USEMMAP`.

—**version** Displays the version number of the **mpatrol** command.

mprof command options

—**addresses** Specifies that different call sites from within the same function are to be differentiated and that the names of all functions should be displayed with their call site offset in bytes.

—**call-graph** Specifies that the allocation call graph should be displayed.

—**counts** Specifies that certain tables should be sorted by the number of allocations or deallocations rather than the total number of bytes allocated or deallocated.

—**graph-file** <*file*> Specifies that the allocation call graph should also be written to a graph specification file for later visualisation with **dot**.

—**help** Displays a quick-reference option summary.

—**leaks** Specifies that memory leaks rather than memory allocations are to be written to the graph specification file.

—**stack-depth** <*depth*> Specifies the maximum stack depth to use when calculating if one call site has the same call stack as another call site. This also specifies the maximum number of functions to display in a call stack.

—**version** Displays the version number of the **mprof** command.

mptrace command options

The **mptrace** command can be built with GUI support on certain platforms. Any of the following options that are marked as being specific to the GUI version of **mptrace** will be read by the X command line parser rather than directly by **mptrace**. As a result they are parsed according to X toolkit rules and do not appear in the quick-reference option summary produced by the ——**help** option. The application class for setting **mptrace** X resources is called `MPTrace`.

—**alloc** <*colour*> Specifies the colour to use for displaying allocated memory (GUI only).

—**base** <*address*> Specifies the base address of the visible address space displayed in the memory map (GUI only).

—**delay** <*length*> Specifies that a small delay of a certain length should be added after drawing each memory allocation event (GUI only).

—**free** <*colour*> Specifies the colour to use for displaying free memory (GUI only).

—**gui** Displays the GUI (if supported).

—**height** <*size*> Specifies the height (in pixels) of the drawing area (GUI only).

—**hatf-file** <*file*> Specifies that the trace should also be written to a file in Heap Allocation Trace Format (HATF).

—**help** Displays a quick-reference option summary.

—**internal** <*colour*> Specifies the colour to use for displaying internal heap memory (GUI only).

—**sim-file** <*file*> Specifies that a trace-driven memory allocation simulation program written in C should be written to a file.

—**source** Displays source-level information for each event in the tracing table, if available.

—**space** <*size*> Specifies the size (in megabytes) of the visible address space displayed in the memory map (GUI only).

—**unalloc** <*colour*> Specifies the colour to use for displaying unallocated heap memory (GUI only).

—**view-height** <*size*> Specifies the height (in pixels) of the window (GUI only).

—**view-width** <*size*> Specifies the width (in pixels) of the window (GUI only).

—**width** <*size*> Specifies the width (in pixels) of the drawing area (GUI only).

—**verbose** Specifies that the tracing table should be displayed.

—**version** Displays the version number of the **mptrace** command.

mleak command options

—**help** Displays a quick-reference option summary.

—**ignore** Specifies that the list of unfreed allocations in the log file should be ignored.

—**max-stack** <*depth*> Specifies the maximum stack depth to display.

—**version** Displays the version number of the **mleak** command.

mpsym command options

—**help** Displays a quick-reference option summary.

—**skip** Skip symbols marked as `???` in the log file.

—**version** Displays the version number of the **mpsym** command.

mpedit command options

The **mpedit** command recognises the `EDITOR` environment variable which can be used to specify the text editor that it should use to edit source files. It also recognises the `MPATROL_SOURCEPATH` environment variable which can be used to specify a colon-separated list of directories that should be used to help search for source files.

—**editor** <*filename*> Specifies the text editor to use.

—**help** Displays a quick-reference option summary.

—**listing** Displays a context listing of the source line instead of invoking the text editor.

—**source-dir** <*directory*> Adds a directory to the search path used to locate the source file.

—**version** Displays the version number of the **mpedit** command.

hexwords command options

—**help** Displays a quick-reference option summary.

—**match** <*exact*|*lower*|*upper*|*any*> Sets the type of case-sensitivity to use.

—**maximum** <*count*> Sets the maximum number of letters to match.

—**minimum** <*count*> Sets the minimum number of letters to match.

—**version** Displays the version number of the **hexwords** command.