The following rewrites Sections 7.22 (General Utilities <stdlib.h>) and 7.22.3 (Memory management functions) of the C1X draft standard (June 2010) which is available at <http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1494.pdf>. Your suggestions on this proposal are most welcome.

**DEADLINE IS 15th OCTOBER 2010**

Big questions for this proposal still to be answered:

1. The type attribute of object alignment is one of the major features of the C1X standard (see Section 6.2.8). Given its pervasiveness throughout statically declared object storage, it would be more consistent if dynamically allocated storage also kept its alignment across calls to realloc without having to respecify it in the arguments as the API below does. Given that alignment ought to be a two’s power, this requires perhaps an extra four bits of storage per block allocated in the memory allocator implementation. Is this too great an overhead for a feature which is not *presently* used to a great extent? Note that with increasing vectorisation and stream computing CPU technologies, the chances are that alignment is going to become much more important in the near future.
2. How far ought we to go in adding introspection routines which allow the querying of the layout of virtual address space? Such querying isn’t of much use unless one can also force the allocator to try to use a certain address for an allocation.
3. Should we also add independent\_malloc and independent\_comalloc? These functions allocate a sequence (of same sized and differently sized blocks respectively) where that sequence is guaranteed to be consecutive in memory, but where each block can be thereafter treated independently. This can greatly improve cache locality for certain kinds of code implementation, and it may be a worthy addition as cache locality becomes ever more important in modern architectures.

Notes on this proposal:

* Separate memory pools have been **deliberately** left out. Rationales:
  1. There are plenty of existing third party memory allocators which already do this.
  2. There are significant changes coming in the next decade in memory allocator implementation theory, and the ISO C specification is not the place to predict these developments. The exponentially rising number of CPU cores in processors will surely additionally introduce elements of non-unified and virtualized memory allocation theory, and one can see how separate memory pools could be an elegant solution to this problem.
* One will surely note how the extra parameters, when defaulted to zero, equal the original C function. It is expected that malloc, calloc, realloc and free simply become parameter defaulting wrappers around the new functions – and equally, that on C++ the extra parameters are given a default of zero. If C ever gets default parameters then this case is covered.

**7.22 General utilities** <stdlib.h>

1. The header <stdlib.h>declares five types and several functions of general utility, and defines several macros.
2. The types declared are size\_tand wchar\_t(both described in 7.19),

div\_t

which is a structure type that is the type of the value returned by the divfunction,

ldiv\_t

which is a structure type that is the type of the value returned by the ldivfunction, and

lldiv\_t

which is a structure type that is the type of the value returned by the lldivfunction.

1. The macros defined are NULL(described in 7.19);

EXIT\_FAILURE

and

EXIT\_SUCCESS

which expand to integer constant expressions that can be used as the argument to the **exit** function to return unsuccessful or successful termination status, respectively, to the host environment;

RAND\_MAX

which expands to an integer constant expression that is the maximum value returned by the randfunction;

MB\_CUR\_MAX

which expands to a positive integer expression with type size\_tthat is the maximum number of bytes in a multibyte character for the extended character set specified by the current locale (category LC\_CTYPE), which is never greater than MB\_LEN\_MAX;

M2\_ZERO\_MEMORY

which requests malloc2 and realloc2 to return any newly allocated space initialized to all bits zero;

M2\_PREVENT\_MOVE

which inhibits the address relocation of an object being resized via realloc2;

M2\_CONSTANT\_TIME

which *may* cause the implementation to avoid any atypical (e.g. housekeeping) operations taking an unpredictable length of time during this particular operation;

M2\_RESERVE\_MULT(N)

which *may* cause malloc2 and realloc2 to reserve N times the initial allocation size of address space, thus allowing subsequent object size expansions via realloc2 up to that reservation to not relocate that object in memory; and

M2\_RESERVE\_SHIFT(N)

which *may* cause malloc2 and realloc2 to reserve 2N bytes of address space for this object, thus allowing subsequent object size expansions via realloc2 up to that reservation to not relocate that object in memory, with this setting overriding M2\_RESERVE\_MULT(N) if also present.

**7.22.3 Memory management functions**

1. The order and contiguity of storage allocated by successive calls to the calloc, malloc, malloc2, realloc and realloc2functions is unspecified. The pointer, unless otherwise specified by a non-zero value for an alignment parameter, returned if the allocation succeeds is *default aligned*, which is defined as the alignment such that it may be assigned to a pointer to any type of object with a fundamental alignment requirement and then used to access such an object or an array of such objects in the space allocated (until the space is explicitly deallocated). The lifetime of an allocated object extends from the allocation until the deallocation. Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer returned points to the start (lowest byte address) of the allocated space. If the space cannot be allocated according to the values of the parameters supplied, a null pointer is returned. If the size of the space requested is zero, the behavior is implementation-defined: either a null pointer is returned, or the behavior is as if the size were some nonzero value, except that the returned pointer shall not be used to access an object.
2. If there is a flags parameter taken by the call, this consists of a bitwise addition (i.e. operator |) of the following macro defined flags as defined at the top of Section 7.22:

* M2\_ZERO\_MEMORY
* M2\_PREVENT\_MOVE
* M2\_CONSTANT\_TIME
* M2\_RESERVE\_MULT(N)
* M2\_RESERVE\_SHIFT(N)

It is expected that implementations may add additional flags not specified here. The right to use up to half the bits provided by a uintmax\_t in future versions of this specification is reserved.

**7.22.3.1 The calloc function**

**Synopsis**

1. #include <stdlib.h>

void \*calloc(size\_t nmemb, size\_t size);

**Description**

1. The callocfunction allocates space for an array of nmembobjects, each of whose size is size. The space is initialized to all bits zero.

**Returns**

1. The callocfunction returns either a null pointer or a pointer to the allocated space.

**7.22.3.2 The free function**

**Synopsis**

1. #include <stdlib.h>

void free(void \*ptr);

**Description**

1. The freefunction causes the space pointed to by ptrto be deallocated, that is, made available for further allocation. If ptris a null pointer, no action occurs. Otherwise, if the argument does not match a pointer earlier returned by a memory management function, or if the space has been deallocated by a call to the free, free2, realloc or realloc2 function, the behavior is undefined.

**Returns**

1. The freefunction returns no value.

**7.22.3.3 The free2 function**

**Synopsis**

1. #include <stdlib.h>

void free2(void \*ptr, uintmax\_t flags);

**Description**

1. The free2function causes the space pointed to by ptrto be deallocated, that is, made available for further allocation. If ptris a null pointer, no action occurs. Otherwise, if the argument does not match a pointer earlier returned by a memory management function, or if the space has been deallocated by a call to the free, free2, realloc or realloc2 function, the behavior is undefined.

**Returns**

1. The free2function returns no value.

**7.22.3.4 The malloc function**

**Synopsis**

1. #include <stdlib.h>

void \*malloc(size\_t size);

**Description**

1. The mallocfunction allocates space for an object whose size is specified by **size** and whose value is indeterminate.

**Returns**

1. The mallocfunction returns either a null pointer or a pointer to the allocated space.

**7.22.3.5 The malloc2 function**

**Synopsis**

1. #include <stdlib.h>

void \*malloc2(size\_t size, size\_t alignment, uintmax\_t flags);

**Description**

1. The malloc2function allocates space for an object whose alignment is specified by **alignment** if that parameter is non-zero, whose size is specified by **size**, and whose value is indeterminate unless the flag M2\_ZERO\_MEMORY is specified. If non-zero, the value of **alignment** shall be a valid alignment supported by the implementation and the value of **size** shall be an integral multiple of **alignment**.

**Returns**

1. The malloc2function returns either a null pointer or a pointer to the allocated space.

**7.22.3.6 The malloc\_usable\_size function**

**Synopsis**

1. #include <stdlib.h>

size\_t malloc\_usable\_size(void \*ptr);

**Description**

1. The malloc\_usable\_sizefunction returns the size in bytes of the object pointed to by ptr. This will always be no smaller than the size requested when allocating or resizing the object, however it may or may not be larger.
2. If ptrdoes not match a pointer earlier returned by a memory management function, or if the space has been deallocated by a call to the free, free2, realloc or realloc2 function, the behavior is undefined.

**Returns**

1. The malloc\_usable\_sizefunction returns the usable space in bytes of the object pointed to by ptr.

**7.22.3.7 The realloc function**

**Synopsis**

1. #include <stdlib.h>

void \*realloc(void \*ptr, size\_t size);

**Description**

1. The reallocfunction deallocates the old object pointed to by ptrand returns a pointer to a new object that has the size specified by size. The contents of the new object shall be the same as that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in the new object beyond the size of the old object have indeterminate values.
2. If ptris a null pointer, the reallocfunction behaves like the mallocfunction for the specified size. Otherwise, if ptrdoes not match a pointer earlier returned by a memory management function, or if the space has been deallocated by a call to the free, free2, realloc or realloc2 function, the behavior is undefined. If memory for the new object cannot be allocated, the old object is not deallocated and its value is unchanged.

**Returns**

1. The reallocfunction returns a pointer to the new object (which may have the same value as a pointer to the old object), or a null pointer if the new object could not be allocated.

**7.22.3.8 The realloc2 function**

**Synopsis**

1. #include <stdlib.h>

void \*realloc2(void \*ptr, size\_t size, size\_t alignment, uintmax\_t flags);

**Description**

1. The realloc2function deallocates the old object pointed to by ptrand returns a pointer to a new object that has the size specified by size and whose alignment is specified by **alignment** if that parameter is non-zero. The contents of the new object shall be the same as that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in the new object beyond the size of the old object have indeterminate values unless flags contains M2\_ZERO\_MEMORY, in which case the newly allocated bytes are initialized to all bits zero.
2. If non-zero, the value of **alignment** shall be a valid alignment supported by the implementation and the value of **size** shall be an integral multiple of **alignment**. If the value of alignment differs from the value of alignment as specified when the old object pointed to by ptr was last allocated or resized, the behavior is undefined.
3. If ptris a null pointer, the realloc2function behaves like the malloc2function for the specified size. Otherwise, if ptrdoes not match a pointer earlier returned by a memory management function, or if the space has been deallocated by a call to the free, free2, realloc or realloc2 function, the behavior is undefined.
4. If ptr is non-zero and flags contains M2\_PREVENT\_MOVE, the pointer returned is guaranteed to either be ptr or a null pointer if the new object could not be allocated. If ptr is a null pointer and flags contains M2\_PREVENT\_MOVE, the realloc2 function behaves like the malloc2 function.
5. If memory for the new object cannot be allocated, the old object is not deallocated and its value is unchanged.

**Returns**

1. The realloc2function returns a pointer to the new object (which will have the same value as a pointer to the old object if M2\_PREVENT\_MOVE was specified, and may have the same value as a pointer to the old object otherwise), or a null pointer if the new object could not be allocated.