OpenClovis
Software Development Kit (SDK)
Service Description and API Reference for Heap Management Service

For OpenClovis SDK Release 2.3 V0.7

Document Revision Date: March 27, 2007

Contents

Chapter 1

Functional Overview

Computer programs use dynamic memory allocation to access memory area whose size is not known at compile time. The **OpenClovis heap library** provides the ability to optimize the performance and reduce fragmentation of free memory. ASP components and applications built on ASP can use this implementation for their dynamic memory requirements.

Applications can take ownership of a certain amount of memory using the `clHeapAllocate`, `clHeapCalloc` or `clHeapRealloc` functions. The ownership of the memory chunk remains with the application until it is explicitly freed by invoking `clHeapFree` API. After the memory is freed, it can be used for re-allocation.

Using the heap library, an application can specify any of the following to be used for dynamic memory allocation:

- Native 'C' implementation
- OpenClovis implementation
- Any other implementation

If the OpenClovis implementation is used, the heap library creates pools of memory chunks of one size. Thus different pools for different chunk sizes can be created within the same heap library. Each pool is created with an initial size that can later grow until an upper limit on the size of that pool or a process wide upper limit on dynamic memory is reached.

The pools grow in quantum of `incrementPoolSize` parameter. The size can be configured in the heap library during its initialization using `clHeapLibInitialize` function. The library must be initialized before any call for dynamic memory allocation is made. When the process is terminated, the heap library must be finalized using `clHeapLibFinalize` function. After finalizing the call, no dynamic memory allocation or de-allocation of memory should be attempted. The OpenClovis implementation also provides Notifications for conditions when the process limit crosses certain values. This enables the resource managers to take appropriate action.

OpenClovis implementation of the heap utility helps the application developer in detecting certain dynamic memory related issues. For example, detection of double free of a chunk, writing over the upper limit of a chunk (overrun), or writing below the lower limit of a chunk.

These facilities are provided by setting the debug level of the library. It can also detect free memory chunk that is not allocated from the heap library or any attempt to free a partial chunk. An application can provide its own implementation of dynamic memory allocation using `CL_HEAP_CUSTOM_MODE` while invoking the `clHeapLibInitialize` API. It can plug-in its own functions for allocation, re-allocation, and de-allocation of memory. All ASP components linked to this application can use the application registered functions for their own requirements.

Chapter 2

Service APIs

2.1 Type Definitions

2.1.1 ClHeapModeT

Header File:
clHeapApi.h

typedef enum {
CL_HEAP_PREALLOCATED_MODE,
CL_HEAP_NATIVE_MODE,
CL_HEAP_CUSTOM_MODE,
} ClHeapModeT;

The enumeration, `ClHeapModeT`, contains the heap allocation modes. The attributes of this enumeration are:

- *CL_HEAP_PREALLOCATED_MODE* - OpenClovis implementation of the memory management library.
- *CL_HEAP_NATIVE_MODE* - Native C mode. It maps to the memory APIs provided by `libc`.
- *CL_HEAP_CUSTOM_MODE* - Custom pools. The application developer can plug in customized memory management library calls.

2.1.2 ClHeapConfigT

Header File:
clHeapApi.h

typedef struct {
ClHeapModeT mode;
ClBoolT lazy;
*ClPoolConfigT *pPoolConfig;*
ClUint32T numPools;
} ClHeapConfigT;

The structure, ClHeapConfigT, contains the configuration of the heap library. The attributes of this structure are:

- *mode* - Allocation mode. This can be either CL_HEAP_NATIVE_MODE, CL_HEAP_PREALLOCATED_MODE, or CL_HEAP_CUSTOM_MODE.
- *lazy* - A pool can grow even after it exhausts its current allocation. This attribute configures a pool in lazy mode for pool expansion. There are two modes for configuring the pool:
  - Lazy mode - The incremented pool does not initialize until an allocation is made from the extended portion of the pool. Lazy mode speeds up the initialization of the application, but shifts the penalty of pool initialization to a later allocation.
  - Normal mode- The incremented pool initializes as soon as it is acquired by the memory management library during the creation of the pool.
- *pPoolConfig* - Array of pool configurations.
- *numPools* - Number of pools in the pPoolConfig array.

2.1.3 ClPoolConfigT

Header File:
clPoolIpi.h

```
typedef struct {
        ClUint32T chunkSize;
        ClUint32T initialPoolSize;
        ClUint32T incrementPoolSize;
        ClUint32T maxPoolSize;
} ClPoolConfigT;
```

The structure, ClPoolConfigT, contains the configuration for a single memory pool. A pool is a collection of memory chunks of same size as mentioned in the chunkSize parameter. The attributes of this structure are:

- *chunkSize* - Size of each memory chunk in this pool specified in bytes.
- *initialPoolSize* - Initial size of this pool in bytes. This pool is created during clHeapLibInitialize. It is a multiple of incrementPoolSize.
- *incrementPoolSize* - When a pool exhausts all its memory chunks, it can grow by this amount to create new memory chunks, which can be allocated on subsequent requests. This size is specified in bytes and is a multiple of chunkSize.
- *maxPoolSize* - Maximum size of the pool in bytes. The pool must not grow beyond this size. Allocation requests from this pool will fail after the pool reaches this size and no memory chunk is available to be allocated. This is a multiple of incrementPoolSize.

2.1.4 ClPoolShrinkFlagsT

Header File:
clPoolIpi.h

typedef enum ClGmsTrackFlags {
CL_pool_SHRINK_DEFAULT,
CL_pool_SHRINK_ONE,
CL_pool_SHRINK_ALL
} ClPoolShrinkFlagsT;
The enumeration, `ClPoolShrinkFlagsT`, indicates the choice you can make while shrinking a pool. For the purpose of shrinking, a concept of extended pool is defined. One extended pool is a memory block of `incrementPoolSize` for that pool. The attributes of this enumeration are:

- *CL_pool_SHRINK_DEFAULT* - Shrinks the pool so that half of the free extended pools are released. Memory chunk is not allocated from a free extended pool.
- *CL_pool_SHRINK_ONE* - Shrinks the pool by one extended pool, if a free extended pool is available.
- *CL_pool_SHRINK_ALL* - Shrinks the pool so that the free extended pools are released.

2.1.5 ClPoolShrinkOptionsT

Header File:
clPoolIpi.h

typedef struct {
ClPoolShrinkFlagsT shrinkFlags;
} ClPoolShrinkOptionsT;

The structure, `ClPoolShrinkOptionsT`, is used to specify the shrink options to the heap library.

- *shrinkFlags* - Specifies the extent to which a pool can shrink.

2.1.6 ClMemStatsT

typedef struct {
ClUint32T numAllocs;
ClUint32T numFrees;
ClUint32T currentAllocSize;
ClUint32T maxAllocSize;
ClUint32T numPools;
} ClMemStatsT;

The structure, `ClMemStatsT`, retrieves statistics from the heap library. The attributes of this structure are:

- *numAllocs* - Total number of allocations (including `clHeapAllocate`, `clHeapCalloc` and `clHeapRealloc`) since the initialization of heap library.
- *numFrees* - Specifies the number of frees (including `clHeapFree` and `clHeapRealloc` APIs) since the initialization of heap library.
- *currentAllocSize* - Specifies the total number of bytes currently allocated by heap library.

- *maxAllocSize* - Specifies the maximum value of `currentAllocSize` as the initialization of heap library. It implies the maximum demand of memory (in bytes) experienced by this process.
- *numPools* - Specifies the total number of pools created by heap library. This value is supplied when `clHeapLibInitialize` is invoked.

2.1.7 ClPoolStatsT

Header File:
clPoolIpi.h

```
typedef struct {
        ClPoolConfigT poolConfig;
        ClUint32T numExtendedPools;
        ClUint32T maxNumExtendedPools;
        ClUint32T numAllocs;
        ClUint32T numFrees;
        ClUint32T maxNumAllocs;
} ClPoolStatsT;
```

The structure, `ClPoolStatsT`, is used to retrieve the statistics from the heap library for a pool of memory chunks.

- *poolConfig* - Configuration of this pool supplied during initialization of this library.
- *numExtendedPools* - Current number of extended pools. One extended pool is a memory block of `incrementPoolSize` for the pool.
- *maxNumExtendedPools* - Maximum value of `numExtendedPools` since the creation of the pool.
- *numAllocs* - Total number of allocations from this pool since the creation of the pool.
- *numFrees* - Total number of frees to this pool since the creation of this pool.
- *maxNumAllocs* - Maximum number of outstanding allocations of this pool. These outstanding allocations are those that have not been freed since the creation of the pool.

2.2 library Life Cycle APIs

2.2.1 clHeapLibInitialize

clHeapLibInitialize

Synopsis:
Initializes the heap library.

Header File:
clHeapApi.h

Syntax:

```
ClRcT clHeapLibInitialize(
                                        CL_IN const ClHeapConfigT* pConfig);
```

Parameters:
pConfig: (in) Pointer to the configuration to be used by heap library.

Return values:
CL_OK: The API executed successfully.
CL_ERR_NULL_POINTER: `pConfig` or `pPoolConfig` (member of `pConfig`) is NULL.
CL_ERR_INVALID_PARAMETER: `numPools` (member of `pConfig`) is 0.
CL_ERR_NO_MEMORY: Heap library is out of memory and cannot proceed further.

Description:
This function is used to initialize the heap library. It is called during the initialization of the EO (Execution Object). The heap library must be initialized before it can be used to allocate memory. The caller should allocate or free the `pConfig` parameter. Any configuration provided through this function to the heap library cannot be changed while the process calling this function is executing.
During the lifetime of a process, this function must be called only once, preferably during process initialization. Subsequent calls to this function are ignored and it returns `CL_OK` without changing anything.

library File:
libClUtils

Related Function(s):
clHeapLibFinalize

2.2.2 clHeapLibFinalize

clHeapLibFinalize

Synopsis:
Finalizes the heap library.

Header File:
clHeapApi.h

Syntax:

```
ClRcT clHeapLibFinalize(void);
```

Parameters:
None.

Return values:
CL_OK: The API executed successfully.

CL_ERR_NOT_INITIALIZED: heap library is not initialized through a previous call to `clHeapLibInitialize` or it is finalized through a call to `clHeapLibFinalize`.

Description:
This function is used to finalize the heap library. After finalizing the heap library, it must not be used to allocate any memory or free previously allocated memory.

library File:
libClUtils

Related Function(s):
clHeapLibInitialize

2.3 Functional APIs

2.3.1 clHeapAllocate

clheapAllocate

Synopsis:
Allocates memory of the requested size.

Header File:
clHeapApi.h

Syntax:

```
void* clHeapAllocate(
                                CL_IN ClUint32T size);
```

Parameters:
size (in): Number of memory bytes to be allocated.

Return values:
The function returns a valid pointer on success and returns NULL on memory allocation failure.

Description:
This function allocates memory of a specified size. The returned memory is aligned at an 8-byte boundary. When the heap library is configured to `CL_HEAP_PREALLOCATED_MODE`, it returns memory of minimum size. This memory size is greater than or equal to the requested size. If size is specified as 0, the system returns a valid pointer pointing to a chunk of minimum size that is previously configured. If heap library is configured to `CL_HEAP_PREALLOCATED_MODE`, failure of `clHeapAllocate` for one size of memory does not mean that this function will fail for other sizes. For more information, refer to man page of malloc(3).

library File:
libClUtils

Related Function(s):
clHeapCalloc, clHeapFree, clHeapRealloc

2.3.2 clHeapFree

clheapFree

Synopsis:
Frees a pre-allocated memory.

Header File:
clHeapApi.h

Syntax:

```
void clHeapFree(
                                    CL_IN void* pAddress);
```

Parameters:
pAddress (in): Block of memory to be freed.

Return values:
None.

Description:
This function is used to free memory. `pAddress` should be a valid pointer allocated through a previous call to either `clHeapAllocate`, `clHeapRealloc`, or `clHeapCalloc`. `pAddress` should not be used after a call to `clHeapFree`. NULL is a valid value for `pAddress`. For more information, refer to man page of free(3).

library File:
libClUtils

Related Function(s):
clHeapCalloc, clHeapAllocate, clHeapRealloc

2.3.3 clHeapCalloc

clheapCalloc

Synopsis:
Allocates memory for an array and initializes it to zero.

Header File:
clHeapApi.h

Syntax:

```
void* clHeapCalloc(
                CL_IN ClUint32T numChunks,
                      CL_IN ClUint32T chunkSize);
```

Parameters:
numChunks (in): Number of chunks to be allocated.

chunkSize (in): Size of each chunk.

Return values:
The function returns a valid pointer on success and returns NULL on memory allocation failure.

Description:
This function allocates memory of a specific size. The memory chunk, it returns, is aligned at an 8-byte boundary. If `CL_HEAP_PREALLOCATED_MODE` is selected during heap configuration, failure of `clHeapAllocate` for one size of memory does not mean that it will fail for other sizes also. For more information, refer to man page, `malloc(3)`. Also refer to man page of `calloc(3)`.

library File:
libClUtils

Related Function(s):
clHeapAllocate, clHeapFree, clHeapRealloc

2.3.4 clHeapRealloc

clheapRealloc

Synopsis:
Changes the size of the memory block (chunk).

Header File:
clHeapApi.h

Syntax:

```
void *clHeapRealloc(
    CL_IN void *pAddress,
    CL_IN ClUint32T size);
```

Parameters:
pAddress: (in) Original pointer to the memory block (chunk).

size: (in) New size of the memory block (chunk).

Return values:
The function returns a valid pointer on success and returns NULL on memory allocation failure.

Description:
This function is used to change the size of the memory block pointed by `pAddress` to `size` in bytes. The new address returned is aligned at an 8-byte boundary. The contents of the returned memory block is unchanged to the minimum of the old and new sizes. The contents of the memory over the size of the previous block of memory is un initialized.

- If `pAddress` is NULL, the call is equivalent to `clHeapAllocate` (size).
- If `size` is zero, the call is equivalent to `clHeapFree(pAddress)` and the function returns NULL.
- If `pAddress` is NULL and size is zero, it is still equal to `clHeapAllocate (0)`.
- If `pAddress` is not NULL, it must have been returned by an earlier call to `clHeapAllocate/clHeapRealloc/clHeapCalloc`.

If `clHeapRealloc` fails, the original memory block remains untouched (if is not freed or moved). If `clHeapRealloc` succeeds, `pAddress` should no longer be used. For more information, refer to man page of `realloc(3)`.

library File:
libClUtils

Related Function(s):
clHeapAllocate, clHeapFree, clHeapCalloc

2.3.5 clHeapShrink

clheapShrink

Synopsis:
Shrinks the pools of pre-allocated memory to enable other pools to grow.

Header File:
clHeapApi.h

Syntax:

```
ClRcT clHeapShrink(
    CL_IN const ClPoolShrinkOptionsT *pShrinkOptions);
```

Parameters:
pShrinkOptions: (in) `shrinkFlags`, a member of this structure indicates to what extent the existing pools can be shrunk.

Return values:
CL_OK: Function completed successfully.

CL_ERR_NOT_INITIALIZED: The heap library is not initialized by a previous call to `clHeapInitialize`.

Description:
This function is used to shrink the memory used by the pool of one `chunkSize` so that pools of other `chunkSize` can grow without violating the wide upper limit of the process. These shrink options are used for all pools of heap library.

library File:
libClUtils

Related Function(s):
None.

2.3.6 clHeapModeGet

clheapModeGet

Synopsis:
Returns the mode set during configuration.

Header File:
clHeapApi.h

Syntax:
```
ClRcT clHeapModeGet(CL_OUT ClHeapModeT *pMode);
```

Parameters:
pMode: (out) Configuration mode returned by the function.

Return values:
CL_OK: Function completed successfully.

CL_ERR_NOT_INITIALIZED: If this function is called before heap initialization through a `clHeapIntialize()` function.

CL_ERR_NULL_POINTER: If `pMode` is passed as NULL.

Description:
This function returns the configuration mode of the heap in the current process. It can be one of the following values: `CL_HEAP_NATIVE_MODE`, `CL_HEAP_PREALLOCATED_MODE`, or `CL_HEAP_CUSTOM_MODE`.

library File:
libClUtils

Related Function(s):
None.

2.3.7 clHeapStatsGet

clheapStatsGet

Synopsis:
Returns the statistics collected by heap module.

Header File:
clHeapApi.h

Syntax:

```
ClRcT clHeapStatsGet(
    CL_OUT ClMemStatsT *pHeapStats);
```

Parameters:
pHeapStats: (out) Pointer to the memory block where heap module will copy the statistics.

Return values:
CL_OK: Function completed successfully.

CL_ERR_NOT_INITIALIZED: Heap library is not initialized.

CL_ERR_NULL_POINTER: The parameter `pHeapStats` is passed as NULL.

Description:
This function is used by heap to collect the statistics about its usage. Other components can invoke this function to access these statistics.

library File:
libClUtils

Related Function(s):
None.

2.3.8 clHeapPoolStatsGet

clheappoolStatsGet

Synopsis:
Returns the statistics collected by heap library for an individual pool.

Header File:
clHeapApi.h

Syntax:

```
ClRcT clHeapPoolStatsGet(
                CL_IN ClUint32T numPools,
                CL_OUT ClUint32T *pPoolSize,
                CL_OUT ClPoolStatsT *pHeapPoolStats);
```

Parameters:

numpools (in): Number of pools for which the statistics are required. This is used as the size for `pPoolSize` and *pPoolStats*.

ppoolSize (out): Pointer to array which contains the sizes of various pools.

pheappoolStats (out): Pointer to array which contains the statistics of various pools.

Return values:

CL_OK: Function completed successfully.

CL_ERR_NOT_INITIALIZED: This function is called before initializing the heap library using the `clHeapIntialize()` function.

CL_ERR_NULL_POINTER: Either `pPoolSize` or `pPoolStats` or both the parameters is NULL.

Description:
This function is used by components to retrieve statistics about usage of various pools and their current size. The heap module gathers this information.
If `numPools` is less than the number of pools configured in heap, the function returns the size and statistics of the first pool arranged in increasing order of chunk sizes. If `numPools` is greater than the number of pools configured, only first n entries of `pPoolSize` and `pPoolStats` are valid, where `n` is the number of pools configured.

library File:
libClUtils

Related Function(s):
None.

2.3.9 clHeapLibCustomInitialize

clheapLibCustomInitialize

Synopsis:
Customizes the initialization of heap library in `CL_HEAP_CUSTOM_MODE`.

Header File:
clHeapApi.h

Syntax:

```
ClRcT clHeapLibCustomInitialize(
                    CL_IN const ClHeapConfigT* pConfig);
```

Parameters:
pConfig : (in) Pointer to configuration information used by heap library.

Return values:
CL_OK: Function completed successfully.

CL_ERR_NULL_POINTER: Either `pConfig` or `pPoolConfig`, a member of `pConfig` is passed as NULL.

CL_ERR_NO_MEMORY: Heap library is out of memory and cannot proceed further.

Description:
This function is used to customize the initialization of the heap library. This is an open function and the application developer should implement this function.This is called only when an application indicates that it needs to customize heap through `CL_HEAP_CUSTOM_MODE`. The function is available in `ASP/models/<model-name>/config/clHeapCustom.c`. This function must call `clHeapHooksRegister()` with the appropriate function pointers to override the implementation of heap provided by OpenClovis.

library File:
libClUtils

Related Function(s):
clHeapLibInitialize, clHeapLibCustomFinalize

2.3.10 clHeapLibCustomFinalize

clheapLibCustomFinalize

Synopsis:
Customizes the finalization of heap library in `CL_HEAP_CUSTOM_MODE`.

Header File:
clHeapApi.h

Syntax:

```
ClRcT clHeapLibCustomFinialize(void );
```

Parameters:
None.

Return values:
CL_OK: Function completed successfully.

CL_ERR_NOT_INITIALIZED: Heap library is not initialized through a previous call to `clHeapLibInitialize()` or it is finalized using `clHeapLibFinalize()`.

Description:
This function is used to customize the finalization of the heap library. This is an open function and the application developer should implement this function. This is called only when an application indicates that it needs to customize heap through `CL_HEAP_CUSTOM_MODE`. The function is available in `ASP/models/<model-name>/config/clHeapCustom.c` This function must call `clHeapHooksRegister()` with the appropriate function pointers to override the implementation of heap provided by OpenClovis.

library File:
libCIUtils

Related Function(s):
clHeapLibFinalize, clHeapLibCustomInitialize

2.3.11 clHeapHooksRegister

clheapHooksRegister

Synopsis:
Register functions to be used in `CL_HEAP_CUSTOM_MODE`.

Header File:
clHeapApi.h

Syntax:
```
 ClRcT clHeapHooksRegister(
CL_IN void *(*allocHook) (ClUint32T),
                      CL_IN void *(*reallocHook)(void *,ClUint32T),
                      CL_IN void *(*callocHook)(ClUint32T,ClUint32T),
                      CL_IN void (*freeHook)(void *)
                      );
```

Parameters:
allocHook: (in) Function that allocates memory. This function is invoked when an application calls `clHeapAllocate()` function.

reallocHook: (in) Function that changes the size of the memory. This function is invoked when an application calls `clHeapRealloc()` function.

callocHook: (in) Function that allocates memory for an array. This function is invoked when an application calls `clHeapCalloc()` function.

freeHook: (in) Function that frees the memory. This function is invoked when an application calls `clHeapFree()` function.

Return values:
CL_OK: Function completed successfully.

CL_ERR_INITIALIZED: Heap library is not initialized or it is finalized.

CL_ERR_NULL_POINTER: One of the input argument is a NULL pointer.

Description:
In `CL_HEAP_CUSTOM_MODE`, this function is used to register the hooks for memory management. The application should override the default implementation of `clHeapLibCustomInitialize()` and `clHeapLibCustomFinalize()` functions present in `ASP/models/<model-name>/config/clHeapCustom.c`. This function should be called from `clHeapLibCustomInitialize()`. In `CL_HEAP_CUSTOM_MODE`, no memory allocation should be made before registering these functions.
During the life of a process, two different set of hooks for memory allocation should not be used. Unless due care is taken, memory allocated by one set of hooks cannot be freed using the other set of hooks as it leads to corruption of meta data structures.

library File:
libClUtils

Related Function(s):
clHeapHooksDeregister

2.3.12 clHeapHooksDeregister

clheapHooksDeregister

Synopsis:
De-registers the hooks registered for `CL_HEAP_CUSTOM_MODE`.

Header File:
clHeapApi.h

Syntax:

```
ClRcT clHeapHooksDeregister(void);
```

Parameters:
None.

Return values:

CL_OK: Function completed successfully.

CL_ERR_INITIALIZED: Heap library is not finalized through a previous call to the `clHeapLibFinalize()` function.

CL_ERR_NOT_INITIALIZED: Hooks were not registered through a previous call to `clHeapHooksRegister()` function.

Description:
This function is used to De-register the hooks registered for memory management in `CL_HEAP_CUSTOM_MODE`. After a call to this function, the memory related calls cannot be made until another call is made to `clHeapHooksRegister`. This function should be called during the finalization of `clHeapLibFinalize()` through a call to open function `clHeapLibCustomFinalize()` present in `ASP/models/<model-name>/config/clHeapCustom.c`.

library File:
libClUtils

Related Function(s):
clHeapHooksRegister

Chapter 3

Service Management Information Model

TBD

Chapter 4

Service Notifications

TBD

Chapter 5

Configuration

TBD

Chapter 6

Debug CLIs

TBD

Index