Cbject docs

Table of Contents

1. Overview

Cbject makes it easier to write object oriented code in C.

1.1. Features

- Objects
- Classes
- Inheritance
- Polymorphism
- Linked lists

1.2. Usage

Example 1. How to add it to a project

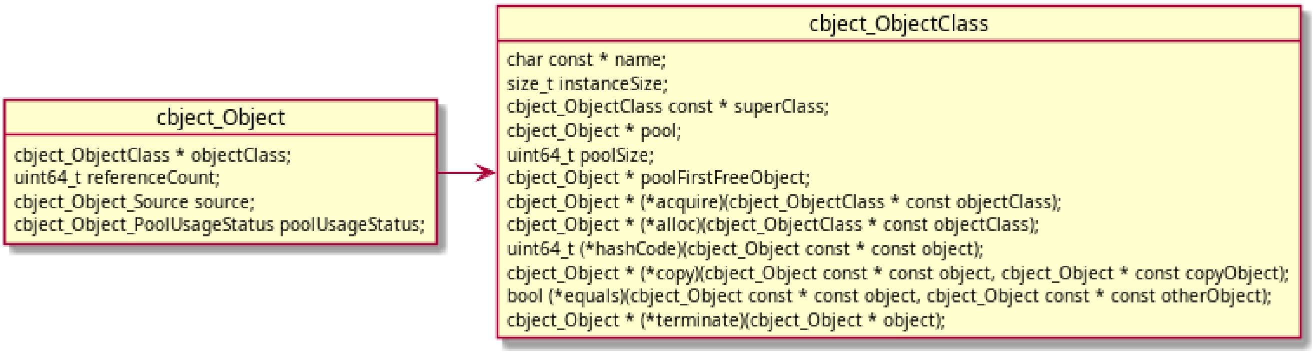
Include the following header file:

```
#include "cbject.h"
```

Example 2. How to create an object

```
cbject_Object * object = cbject_Object_init(cbject_Object_alloc(cbject_Object));
printf("%d\n", cbject_Object_hashCode(object));
cbject_utils_dealloc(object);
```

1.3. cbject_Object model



```
cbject_ObjectClass * objectClass;
uint64_t referenceCount;
cbject_Object_Source source;
cbject_Object_PoolUsageStatus poolUsageStatus;
```

```
char const * name;
size_t instanceSize;
cbject_ObjectClass const * superClass;
cbject_Object * pool;
uint64_t poolSize;
cbject_Object * poolFirstFreeObject;
cbject_Object * (*acquire)(cbject_ObjectClass * const objectClass);
cbject_Object * (*alloc)(cbject_ObjectClass * const objectClass);
uint64_t (*hashCode)(cbject_Object const * const object);
cbject_Object * (*copy)(cbject_Object const * const object, cbject_Object * const copyObject);
bool (*equals)(cbject_Object const * const object, cbject_Object const * const otherObject);
cbject_Object * (*terminate)(cbject_Object * object);
```

Figure 1. Building blocks

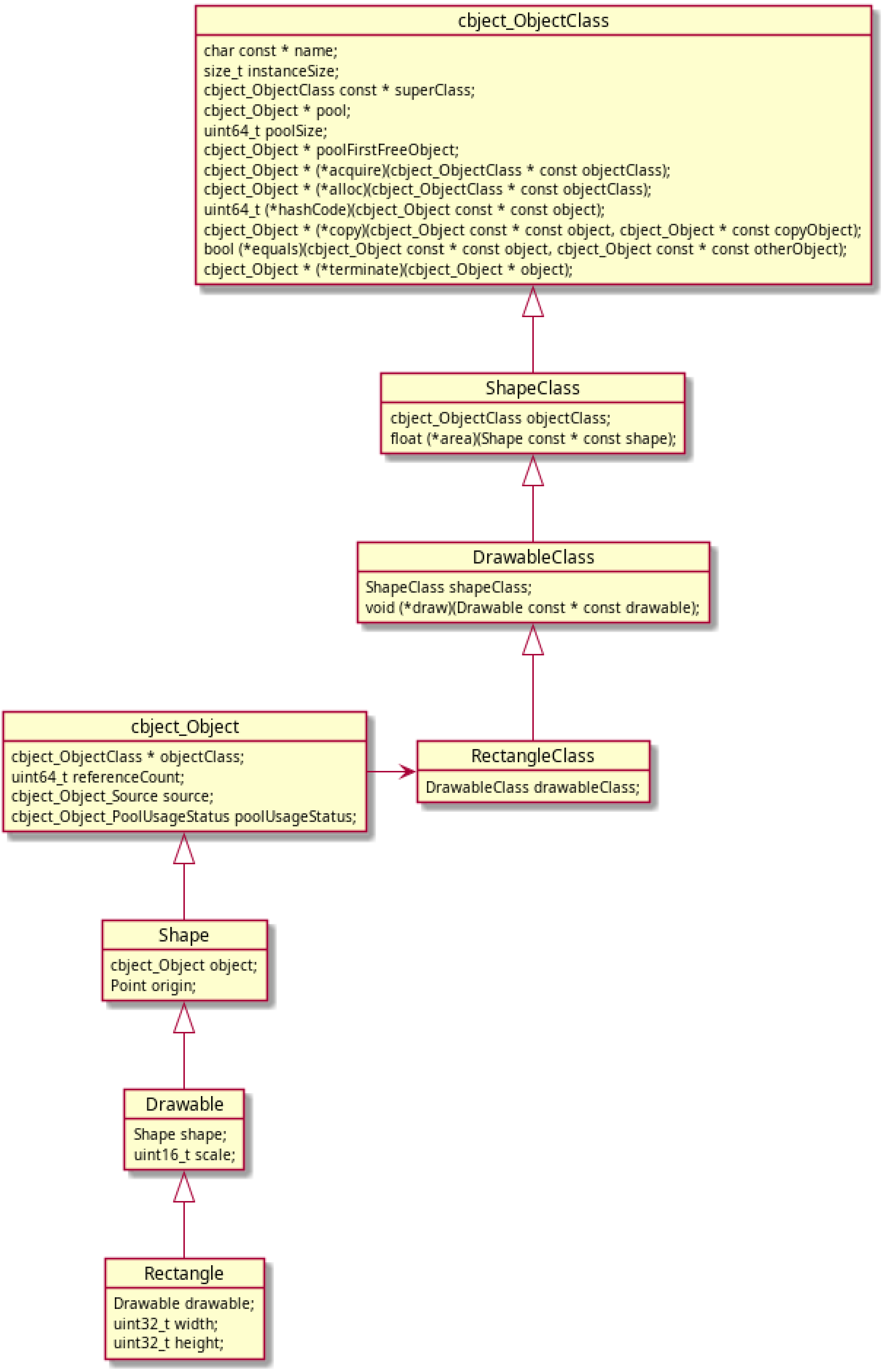

cbject_ObjectClass
char const * name;
size_t instanceSize;
cbject_ObjectClass const * superClass;
cbject_Object * pool;
uint64_t poolSize;
cbject_Object * poolFirstFreeObject;
cbject_Object * (*acquire)(cbject_ObjectClass * const objectClass);
cbject_Object * (*alloc)(cbject_ObjectClass * const objectClass);
uint64_t (*hashCode)(cbject_Object const * const object);
cbject_Object * (*copy)(cbject_Object const * const object, cbject_Object * const copyObject);
bool (*equals)(cbject_Object const * const object, cbject_Object const * const otherObject);
cbject_Object * (*terminate)(cbject_Object * object);
ShapeClass
cbject_ObjectClass objectClass;
float (*area)(Shape const * const shape);
DrawableClass
ShapeClass shapeClass;
void (*draw)(Drawable const * const drawable);
cbject_Object
cbject_ObjectClass * objectClass;
uint64_t referenceCount;
cbject_Object_Source source;
cbject_Object_PoolUsageStatus poolUsageStatus;
RectangleClass
DrawableClass drawableClass;
Shape
cbject_Object object;
Point origin;
Drawable
Shape shape;
uint16_t scale;
Rectangle
Drawable drawable;
uint32_t width;
uint32_t height;

Figure 2. Rectangle class example

2. API

2.1. cbject

2.1.1. Overview

Cbject framework

2.2. cbject_config

2.2.1. Overview

Cbject configuration

2.2.2. Macros

cbject_config_useHeap

```
#define cbject_config_useHeap configValue
```

Heap config

Values

- true
- false

cbject_config_useStaticPool

```
#define cbject_config_useStaticPool configValue
```

Static pool config

Values

- true
- false

cbject_config_useLinkedList

```
#define cbject_config_useLinkedList configValue
```

LinkedList config

Values

- true
- false

cbject_config_linkedListPoolSize

```
#define cbject_config_linkedListPoolSize configValue
```

LinkedList pool size config

Values

- >= 0

cbject_config_useNode

```
#define cbject_config_useNode configValue
```

Node config

Values

- true
- false

cbject_config_nodePoolSize

```
#define cbject_config_nodePoolSize configValue
```

Node pool size config

Values

- >= 0

cbject_config_useSingleton

```
#define cbject_config_useSingleton configValue
```

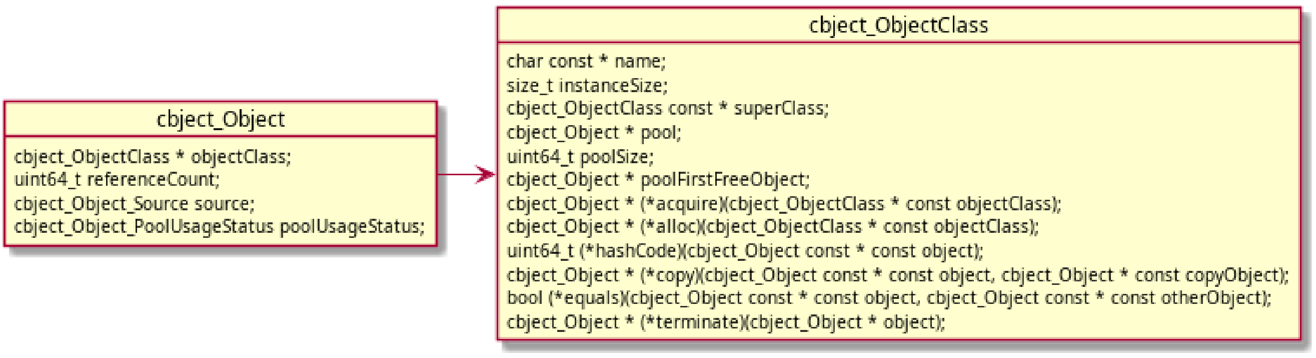
Singleton config

Values

- true
- false

2.3. cbject_Object

2.3.1. Overview



```
cbject_Object
cbject_ObjectClass * objectClass;
uint64_t referenceCount;
cbject_Object_Source source;
cbject_Object_PoolUsageStatus poolUsageStatus;
```

```
cbject_ObjectClass
char const * name;
size_t instanceSize;
cbject_ObjectClass const * superClass;
cbject_Object * pool;
uint64_t poolSize;
cbject_Object * poolFirstFreeObject;
cbject_Object * (*acquire)(cbject_ObjectClass * const objectClass);
cbject_Object * (*alloc)(cbject_ObjectClass * const objectClass);
uint64_t (*hashCode)(cbject_Object const * const object);
cbject_Object * (*copy)(cbject_Object const * const object, cbject_Object * const copyObject);
bool (*equals)(cbject_Object const * const object, cbject_Object const * const otherObject);
cbject_Object * (*terminate)(cbject_Object * object);
```

Figure 3. Context diagram

The building block. All objects defined in Cbject need to extend cbject_Object.

2.3.2. Types

cbject_Object

```
typedef struct cbject_Object cbject_Object;
```

Typedef for struct cbject_Object

cbject_ObjectClass

```
typedef struct cbject_ObjectClass cbject_ObjectClass;
```

Typedef for struct cbject_ObjectClass

cbject_Object_PoolUsageStatus

```
typedef enum {
    cbject_Object_PoolUsageStatus_free = 0,
    cbject_Object_PoolUsageStatus_inUse
} cbject_Object_PoolUsageStatus;
```

Typedef and struct definition for cbject_Object_PoolUsageStatus

Remark

Used for static pool functionality

Values

- free
- inUse

cbject_Object_Source

```
typedef enum {
    cbject_Object_Source_stack,
    cbject_Object_Source_heap,
    cbject_Object_Source_staticPool
} cbject_Object_Source;
```

Typedef and struct definition for cbject_Object_Source

Remark

Used if heap or static pool usage is activated

Values

- free
- inUse

struct cbject_Object

```
struct cbject_Object {
    cbject_ObjectClass * objectClass;
    uint64_t referenceCount;
    cbject_Object_Source source;
    cbject_Object_PoolUsageStatus poolUsageStatus;
};
```

Definition of struct cbject_Object

Members

- objectClass - cbject_ObjectClass reference

- referenceCount - The reference count (number of owners of the object)
- source - Source from where the object was created (stack/heap/staticPool)
- poolUsageStatus - Usage status of object (free/inUse)

struct cbject_ObjectClass

```
struct cbject_ObjectClass {
    char const * name;
    size_t instanceSize;
    cbject_ObjectClass const * superClass;
    cbject_Object * pool;
    uint64_t poolSize;
    cbject_Object * poolFirstFreeObject;
    cbject_Object * (*acquire)(cbject_ObjectClass * const objectClass);
    cbject_Object * (*alloc)(cbject_ObjectClass * const objectClass);
    uint64_t (*hashCode)(cbject_Object const * const object);
    cbject_Object * (*copy)(cbject_Object const * const object, cbject_Object *
const copyObject);
    bool (*equals)(cbject_Object const * const object, cbject_Object const * const
otherObject);
    cbject_Object * (*terminate)(cbject_Object * object);
};
```

Definition of struct cbject_ObjectClass

Members

- name - Name of the class
- instanceSize - Memory size for an instance of the class
- superClass - Super class reference
- pool - Reference to the object static pool
- poolSize - Size of pool (number of objects in pool)
- poolFirstFreeObject - Reference to the first free object in the pool
- acquire - Acquire method reference
- alloc - Alloc method reference
- hashCode - Hash code method reference
- copy - Copy method reference
- equals - Equals method reference
- terminate - Terminate method reference

2.3.3. Functions

cbject_Object_acquire()

```
cbject_Object * cbject_Object_acquire(cbject_ObjectClass * const objectClass);
```

Acquires an object from the static pool

Params

- objectClass - cbject_ObjectClass reference

Return

Reference of the acquired object

cbject_Object_alloc()

```
cbject_Object * cbject_Object_alloc(cbject_ObjectClass * const objectClass);
```

Allocates an object in heap memory

Params

- objectClass - cbject_ObjectClass reference

Return

Reference of the allocated object

cbject_Object_init()

```
cbject_Object * cbject_Object_init(cbject_Object * const object);
```

Initializes an object

Params

- object - cbject_Object reference

Return

Initialized object

cbject_Object_allocHelper()

```
cbject_Object * cbject_Object_allocHelper(cbject_Object * const object,
cbject_ObjectClass * const objectClass);
```

Sets the class of the object and other proprieties needed for allocation

Params

- object - cbject_Object reference
- objectClass - cbject_ObjectClass reference

Return

Reference to the object

cbject_Object_copy()

```
cbject_Object * cbject_Object_copy(cbject_Object const * const object,
cbject_Object * const copyObject);
```

Copies the object to the provided instance.

Params

- object - cbject_Object reference
- copyObject - Reference of a new object in which to copy the original one

Return

Reference of copyObject

cbject_Object_equals()

```
bool cbject_Object_equals(cbject_Object const * const object, cbject_Object const
* const otherObject);
```

Compares two objects

Params

- object - cbject_Object reference
- otherObject - Reference for the compared object

Return

- true - If the objects are equal
- false - If the objects are different

cbject_Object_hashCode()

```
uint64_t cbject_Object_hashCode(cbject_Object const * const object);
```

Gets the hash code of the object

Params

- object - cbject_Object reference

Return

The hash code of the object

cbject_Object_retain()

```
cbject_Object * cbject_Object_retain(cbject_Object * const object);
```

Increases the reference count of the object

Params

- object - cbject_Object reference

Return

Reference to object

cbject_Object_release()

```
void * cbject_Object_release(cbject_Object * const object);
```

Decreases the reference count of the object and performs deallocation if reference count reaches 0

Params

- object - cbject_Object reference

Return

NULL

cbject_Object_isOfClass()

```
bool cbject_Object_isOfClass(cbject_Object const * const object,
cbject_ObjectClass const * const objectClass);
```

Checks if an object is of a given class

Params

- object - cbject_Object reference
- objectClass - Class reference

Return

- true - If the object is of the provided class
- false - If the object is of a different class

cbject_ObjectClass_instance()

```
cbject_ObjectClass * cbject_ObjectClass_instance(void);
```

Gets cbject_ObjectClass instance

Return

Reference of the class instance

2.3.4. Macros

cbject_ObjectClass_setup()

```
cbject_ObjectClass_setup(klass)
```

Populates the class instance

Remark

cbject_Class must be defined before using this macro

Params

- klass - Class reference

cbject_Object_class()

```
cbject_Object_class(object)
```

Gets the class of an object

Params

- object - cbject_Object reference

Return

Class reference

cbject_Object_instanceSize()

```
cbject_Object_instanceSize(object)
```

Gets the size in memory of an object

Params

- object - cbject_Object reference

Return

The size in memory of the object

2.3.5. Tests

test_cbject_ObjectClass

Test setup of ObjectClass

Steps

1. Get ObjectClass instance
2. Check if object size stored in class is equal to the actual object size
3. Check that the function pointers in the class are initialized

test_cbject_Object_init

Test initialization of cbject_Object

Steps

1. Allocate object on stack an initialize it
2. Check if object class points to cbject_ObjectClass instance

test_cbject_Object_equals

Test equals method

Steps

1. Allocate object on stack an initialize it
2. Check if equals method returns true when comparing object to self
3. Allocate another object on stack an initialize it
4. Check if equals method returns false when comparing the two objects

test_cbject_Object_hashCode

Test hashCode method

Steps

1. Allocate object on stack an initialize it
2. Check if hashCode method returns the address in memory of the object

test_cbject_Object_isOfClass

Test isOfClass method

Preconditions

1. Define a dummy TestClass which extends cbject_ObjectClass

Steps

1. Allocate object on stack an initialize it
2. Check if isOfClass method returns true when checked against cbject_Object
3. Check if isOfClass method returns false when checked against Test

test_cbject_Object_copy

Test copy method

Steps

1. Allocate object on stack an initialize it
2. Allocate another object on stack and copy the first object into it
3. Check if the memory sections occupied by the two objects are equal
4. Allocate another object on heap and copy the first object into it
5. Check if the memory sections occupied by the two objects are equal
6. Deallocate the object from the heap memory

2.4. cbject_Singleton

2.4.1. Overview

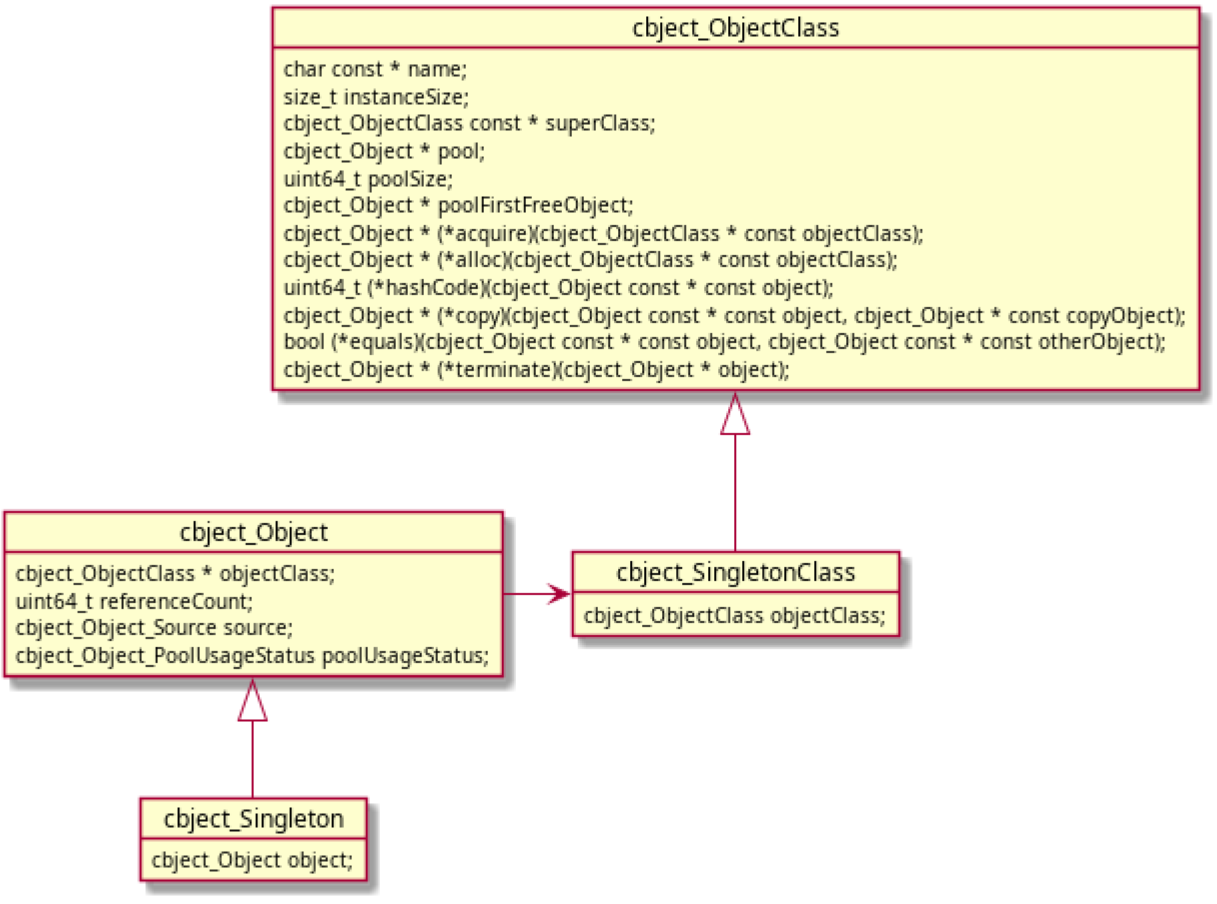

cbject_ObjectClass
char const * name;
size_t instanceSize;
cbject_ObjectClass const * superClass;
cbject_Object * pool;
uint64_t poolSize;
cbject_Object * poolFirstFreeObject;
cbject_Object * (*acquire)(cbject_ObjectClass * const objectClass);
cbject_Object * (*alloc)(cbject_ObjectClass * const objectClass);
uint64_t (*hashCode)(cbject_Object const * const object);
cbject_Object * (*copy)(cbject_Object const * const object, cbject_Object * const copyObject);
bool (*equals)(cbject_Object const * const object, cbject_Object const * const otherObject);
cbject_Object * (*terminate)(cbject_Object * object);
cbject_Object
cbject_ObjectClass * objectClass;
uint64_t referenceCount;
cbject_Object_Source source;
cbject_Object_PoolUsageStatus poolUsageStatus;
cbject_SingletonClass
cbject_ObjectClass objectClass;
cbject_Singleton
cbject_Object object;

Figure 4. Context diagram

2.4.2. Types

cbject_Singleton

```
typedef struct cbject_Singleton cbject_Singleton;
```

Typedef for struct cbject_Singleton

cbject_SingletonClass

```
typedef struct cbject_SingletonClass cbject_SingletonClass;
```

Typedef for struct cbject_SingletonClass

struct cbject_Singleton

```
struct cbject_Singleton {
    cbject_Object object;

};
```

Definition of struct cbject_Singleton

Members

- object - Parent

struct cbject_SingletonClass

```
struct cbject_SingletonClass {
    cbject_ObjectClass objectClass;
};
```

Definition of struct cbject_SingletonClass

Members

- cbject_ObjectCLass - class of parent

2.4.3. Functions

cbject_Singleton_init()

```
cbject_Singleton * cbject_Singleton_init(cbject_Singleton * const singleton);
```

Initializes a singleton

Params

- singleton - cbject_Singleton reference

Return

Initialized singleton

cbject_SingletonClass_instance()

```
cbject_SingletonClass * cbject_SingletonClass_instance(void);
```

Gets cbject_SingletonClass instance

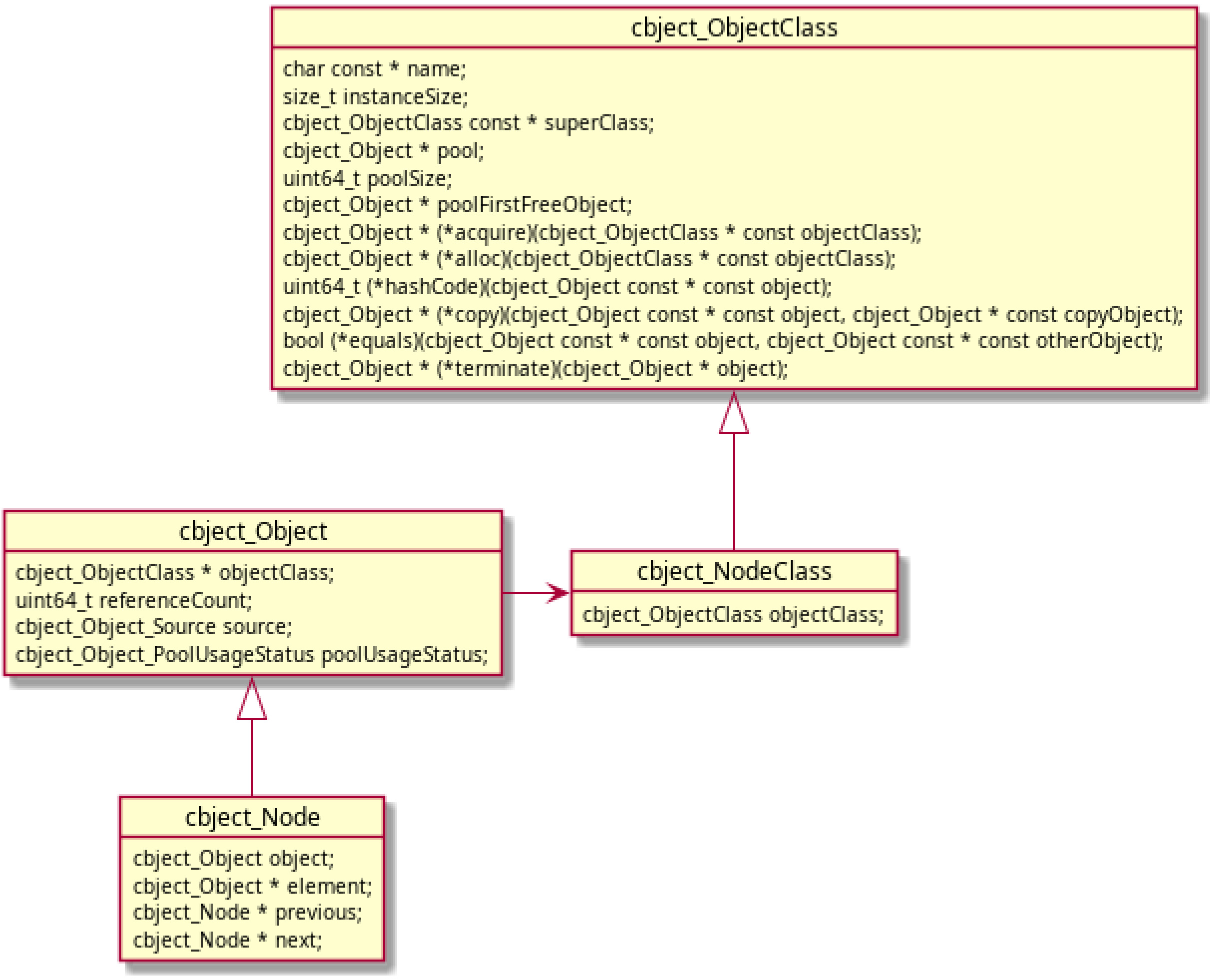
Return

Reference of the class instance

2.5. cbject_Node

2.5.1. Overview

Node data structure used in linked lists



cbject_ObjectClass
char const * name;
size_t instanceSize;
cbject_ObjectClass const * superClass;
cbject_Object * pool;
uint64_t poolSize;
cbject_Object * poolFirstFreeObject;
cbject_Object * (*acquire)(cbject_ObjectClass * const objectClass);
cbject_Object * (*alloc)(cbject_ObjectClass * const objectClass);
uint64_t (*hashCode)(cbject_Object const * const object);
cbject_Object * (*copy)(cbject_Object const * const object, cbject_Object * const copyObject);
bool (*equals)(cbject_Object const * const object, cbject_Object const * const otherObject);
cbject_Object * (*terminate)(cbject_Object * object);
cbject_Object
cbject_ObjectClass * objectClass;
uint64_t referenceCount;
cbject_Object_Source source;
cbject_Object_PoolUsageStatus poolUsageStatus;
cbject_NodeClass
cbject_ObjectClass objectClass;
cbject_Node
cbject_Object object;
cbject_Object * element;
cbject_Node * previous;
cbject_Node * next;

Figure 5. Context diagram

2.5.2. Types

cbject_Node

```
typedef struct cbject_Node cbject_Node;
```

Typedef for struct cbject_Node

cbject_NodeClass

```
typedef struct cbject_NodeClass cbject_NodeClass;
```

Typedef for struct cbject_NodeClass

struct cbject_Node

```
struct cbject_Node {
    cbject_Object object;
    cbject_Object * element;
    cbject_Node * previous;
    cbject_Node * next;

};
```

Definition of struct cbject_Node

Members

- object - Parent
- element - Reference to the element
- previous - Reference to the previous node
- next - Reference to the next node

struct cbject_NodeClass

```
struct cbject_NodeClass {
    cbject_ObjectClass objectClass;
};
```

Definition of struct cbject_NodeClass

Members

- cbject_ObjectCLass - class of parent

2.5.3. Functions

cbject_Node_init()

```
cbject_Node * cbject_Node_init(cbject_Node * const node, cbject_Object * const
object);
```

Initializes a Node

Params

- node - cbject_Node reference
- object - Object to store in the node

Return

Initialized Node

cbject_Node_getElement()

```
cbject_Object * cbject_Node_getElement(cbject_Node const * const node);
```

Gets the data object contained in the node

Params

- node - cbject_Node reference

Return

Data object in the node

cbject_Node_getPrevious()

```
cbject_Node * cbject_Node_getPrevious(cbject_Node const * const node);
```

Gets the previous node

Params

- node - cbject_Node reference

Return

The previous node

cbject_Node_setPrevious()

```
void cbject_Node_setPrevious(cbject_Node * const node, cbject_Node * const
previousNode);
```

Sets the previous node

Params

- node - cbject_Node reference
- previousNode - cbject_Node reference

cbject_Node_getNext()

```
cbject_Node * cbject_Node_getNext(cbject_Node const * const node);
```

Gets the next node

Params

- node - cbject_Node reference

Return

The next node

cbject_Node_setNext()

```
void cbject_Node_setNext(cbject_Node * const node, cbject_Node * const nextNode);
```

Sets the next node

Params

- node - cbject_Node reference
- nextNode - cbject_Node reference

cbject_NodeClass_instance()

```
cbject_NodeClass * cbject_NodeClass_instance(void);
```

Gets cbject_NodeClass instance

Return

Reference of the class instance

2.5.4. Tests

test_cbject_Node_init

Test Node initialization

Steps

1. Create an object and a node which takes the object as input
2. Check node state

test_cbject_Node_setters

Test Node setters

Steps

1. Create 3 nodes (node, previousNode, nextNode)
2. Set previous and next nodes to the first node
3. Check the node state

2.6. cbject_LinkedList

2.6.1. Overview

Linked list data structure

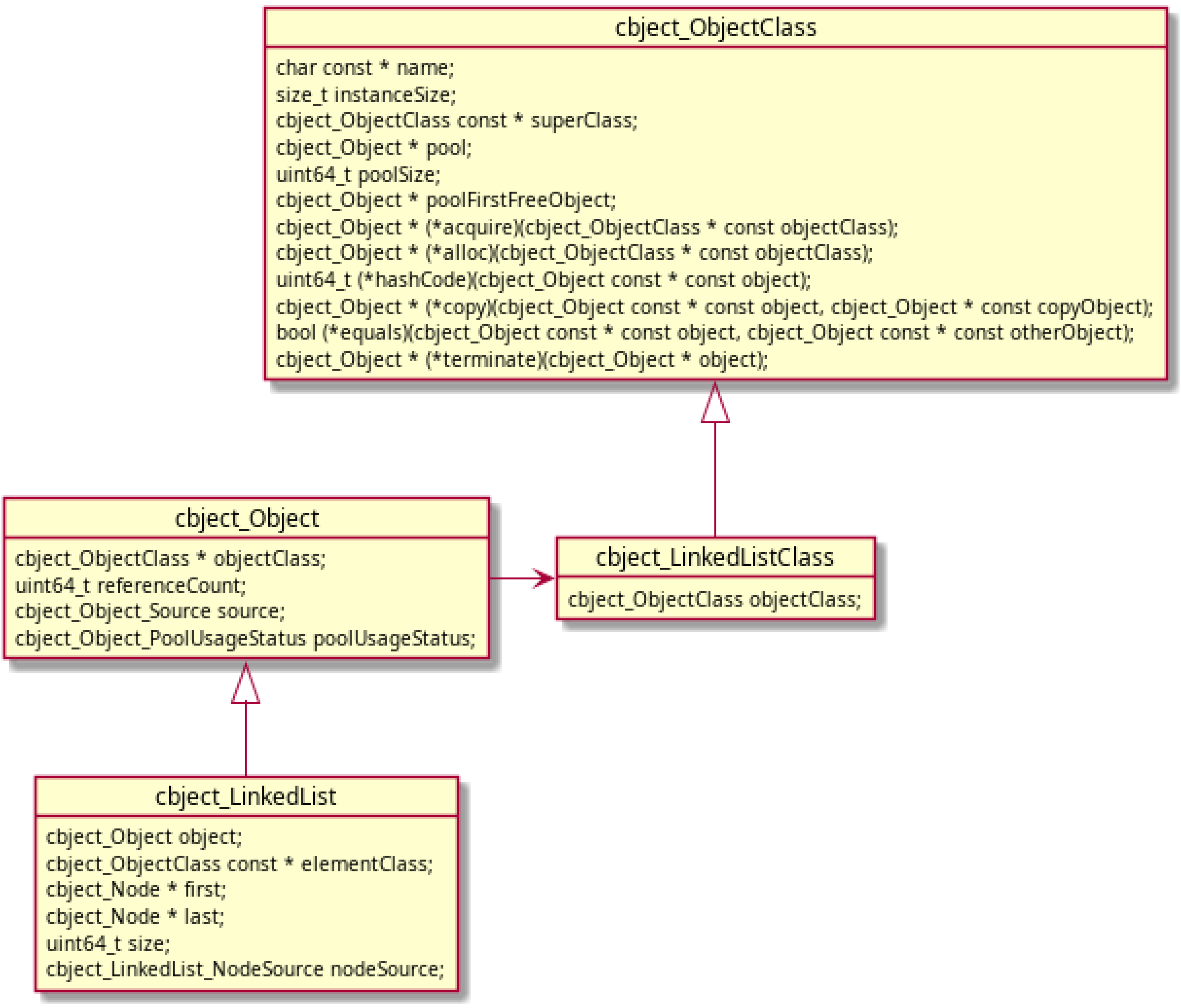

cbject_ObjectClass
char const * name;
size_t instanceSize;
cbject_ObjectClass const * superClass;
cbject_Object * pool;
uint64_t poolSize;
cbject_Object * poolFirstFreeObject;
cbject_Object * (*acquire)(cbject_ObjectClass * const objectClass);
cbject_Object * (*alloc)(cbject_ObjectClass * const objectClass);
uint64_t (*hashCode)(cbject_Object const * const object);
cbject_Object * (*copy)(cbject_Object const * const object, cbject_Object * const copyObject);
bool (*equals)(cbject_Object const * const object, cbject_Object const * const otherObject);
cbject_Object * (*terminate)(cbject_Object * object);
cbject_Object
cbject_ObjectClass * objectClass;
uint64_t referenceCount;
cbject_Object_Source source;
cbject_Object_PoolUsageStatus poolUsageStatus;
cbject_LinkedListClass
cbject_ObjectClass objectClass;
cbject_LinkedList
cbject_Object object;
cbject_ObjectClass const * elementClass;
cbject_Node * first;
cbject_Node * last;
uint64_t size;
cbject_LinkedList_NodeSource nodeSource;

Figure 6. Context diagram

2.6.2. Types

cbject_LinkedList

```
typedef struct cbject_LinkedList cbject_LinkedList;
```

Typedef for struct cbject_LinkedList

cbject_LinkedListClass

```
typedef struct cbject_LinkedListClass cbject_LinkedListClass;
```

Typedef for struct cbject_LinkedListClass

cbject_LinkedList_NodeSource

```
typedef enum {
    cbject_LinkedList_NodeSource_heap,
    cbject_LinkedList_NodeSource_staticPool
} cbject_LinkedList_NodeSource;
```

Typedef and struct definition for cbject_LinkedList_NodeSource

Remark

Used for linked list functionality

Values

- heap
- staticPool

struct cbject_LinkedList

```
struct cbject_LinkedList {
    cbject_Object object;
    cbject_ObjectClass const * elementClass;
    cbject_Node * first;
    cbject_Node * last;
    uint64_t size;
    cbject_LinkedList_NodeSource nodeSource;
};
```

Definition of struct cbject_LinkedList

Members

- object - Parent
- elementClass - Class of the elements stored in the list
- first - Reference to the first node in the list
- last - Reference to the last node in the list
- size - Size of the list (number of elements)
- nodeSource - Source for node creation (heap/staticPool)

struct cbject_LinkedListClass

```
struct cbject_LinkedListClass {
    cbject_ObjectClass objectClass;
};
```

Definition of struct cbject_LinkedListClass

Members

- cbject_ObjectCLass - class of parent

2.6.3. Functions

cbject_LinkedList_init()

```
cbject_LinkedList * cbject_LinkedList_init(
    cbject_LinkedList * const linkedList,
    cbject_LinkedList_NodeSource const nodeSource
);
```

Initializes a LinkedList

Params

- linkedList - cbject_LinkedList reference
- nodeSource - Source for node creation (heap/staticPool) .Return Initialized and empty LinkedList

cbject_LinkedList_isEmpty()

```
bool cbject_LinkedList_isEmpty(cbject_LinkedList const * const linkedList);
```

Checks if list is empty

Params

- linkedList - cbject_LinkedList reference

Return

- true - if list is empty
- false - if list is not empty

cbject_LinkedList_addLast()

```
void cbject_LinkedList_addLast(cbject_LinkedList * const linkedList, cbject_Object
* const object);
```

Adds an element to the end of the list

Params

- linkedList - cbject_LinkedList reference
- object - Object to be added in the list

cbject_LinkedList_addFirst()

```
void cbject_LinkedList_addFirst(cbject_LinkedList * const linkedList,
cbject_Object * const object);
```

Adds an element at the beginning of the list

Params

- linkedList - cbject_LinkedList reference
- object - Object to be added in the list

cbject_LinkedList_removeLast()

```
void cbject_LinkedList_removeLast(cbject_LinkedList * const linkedList);
```

Removes last element in the list

Params

- linkedList - cbject_LinkedList reference

cbject_LinkedList_removeFirst()

```
void cbject_LinkedList_removeFirst(cbject_LinkedList * const linkedList);
```

Removes first element in the list

Params

- linkedList - cbject_LinkedList reference

cbject_LinkedList_clear()

```
void cbject_LinkedList_clear(cbject_LinkedList * const linkedList);
```

Removes all elements from the list

Params

- linkedList - cbject_LinkedList reference

cbject_LinkedList_getFirst()

```
cbject_Object * cbject_LinkedList_getFirst(cbject_LinkedList const * const
linkedList);
```

Gets the first element in the list

Params

- linkedList - cbject_LinkedList reference

Return

First element in list

cbject_LinkedList_getLast()

```
cbject_Object * cbject_LinkedList_getLast(cbject_LinkedList const * const
linkedList);
```

Gets the last element in the list

Params

- linkedList - cbject_LinkedList reference

Return

Last element in list

cbject_LinkedList_get()

```
cbject_Object * cbject_LinkedList_get(cbject_LinkedList const * const linkedList,
uint64_t index);
```

Gets element at specified index

Params

- linkedList - cbject_LinkedList reference
- index - index of the element to return

Return

Element at specified index

cbject_LinkedList_getSize()

```
uint64_t cbject_LinkedList_getSize(cbject_LinkedList const * const linkedList);
```

Gets the size of the list (number of elements)

Params

- linkedList - cbject_LinkedList reference

Return

Size of list (number of elements)

cbject_LinkedListClass_instance()

```
cbject_LinkedListClass * cbject_LinkedListClass_instance(void);
```

Gets cbject_LinkedListClass instance

Return

Reference of the class instance

2.6.4. Tests

test_cbject_LinkedList_init

Test LinkedList initialization

Steps

1. Create a linked list
2. Check class and members
3. Terminate the linked list

test_cbject_LinkedList_addFirst

Test adding elements at beginning of LinkedList

Preconditions

1. Define a DataClass which extends cbject_ObjectClass

Steps

1. Create a linked list and some data objects
2. Add the objects to the list and check the state of the list and the nodes
3. Terminate the linked list

test_cbject_LinkedList_addLast

Test adding elements at the end of LinkedList

Steps

1. Create a linked list and some objects
2. Add the objects to the list and check the state of the list and the nodes
3. Terminate the linked list

test_cbject_LinkedList_removeFirst

Test removing elements at the beginning of the list

Steps

1. Create a linked list and some objects
2. Add the objects to the list, remove them from the list and check the state of the list and the nodes
3. Terminate the linked list

test_cbject_LinkedList_removeLast

Test removing elements at the end of the list

Steps

1. Create a linked list and some objects
2. Add the objects to the list, remove them from the list and check the state of the list and the nodes
3. Terminate the linked list

test_cbject_LinkedList_clear

Test clearing elements from a list

Steps

1. Create a linked list and some objects
2. Add the objects to the list, clear the list and check the state of the list and the nodes
3. Terminate the linked list

2.7. cbject_utils

2.7.1. Overview

TODO

2.7.2. Macros

cbject_utils_acquire()

```
cbject_utils_acquire(klass)
```

Acquires an object from the static pool

Remarks

Calls cbject_Object_acquire() and does the necessary casting

Params

- klass - Name of class

Return

Reference of the acquired object

cbject_utils_alloc()

```
cbject_utils_alloc(klass)
```

Allocates an object in heap memory

Remarks

Calls cbject_Object_alloc() and does the necessary casting

Params

- klass - Name of class

Return

Reference of the allocated object

cbject_utils_stackAlloc()

```
cbject_utils_stackAlloc(klass)
```

Allocates an object on the stack

Params

- klass - Name of class

Return

Reference of the allocated memory

cbject_utils_hashCode()

```
cbject_utils_hashCode(object)
```

Gets the hash code of the object

Remarks

Calls cbject_Object_hashCode() and does the necessary casting

Params

- object - cbject_Object reference

Return

The hash code of the object

cbject_utils_equals()

```
cbject_utils_equals(object, otherObject)
```

Compares two objects

Remarks

Calls cbject_Object_equals() and does the necessary casting

Params

- object - cbject_Object reference
- otherObject - Reference for the compared object

Return

- true - If the objects are equal
- false - If the objects are different

cbject_utils_copy()

```
cbject_utils_copy(object, copyObject)
```

Copies the object to the provided instance.

Remarks

Calls cbject_Object_copy() and does the necessary casting

Params

- object - cbject_Object reference
- copyObject - Reference of a new object in which to copy the original one

Return

Reference of copyObject

cbject_utils_retain()

```
cbject_utils_retain(object)
```

Increases the reference count of the object

Remarks

Calls cbject_Object_retain() and does the necessary casting

Params

- object - cbject_Object reference

Return

Reference to object

cbject_utils_release()

```
cbject_utils_release(object)
```

Decreases the reference count of the object and performs deallocation if reference count reaches 0

Remarks

Calls cbject_Object_release() and does the necessary casting

Params

- object - cbject_Object reference

Return

NULL

cbject_utils_allocPool()

```
cbject_utils_allocPool(poolSize)
```

Allocates a static pool

Remarks

cbject_Class must be defined before using this macro

Params

- poolSize - Size of pool (number of objects in pool)

cbject_utils_doOnce

```
cbject_utils_doOnce
```

Runs a block of code only once

Usage

```
cbject_utils_doOnce {
    functionCall();
    anotherFunctionCall();
}
```

Remark

Not thread safe

cbject_utils_invokeMethod()

```
cbject_utils_invokeMethod(method, ...)
```

Polymorphic call of an object method

Remarks

cbject_Class must be defined before using this macro

Params

- method - Name of the method
- ...
    - object - cbject_Object reference
    - ... - Method params

Return

Depends on the called method

cbject_utils_invokeClassMethod()

```
cbject_utils_invokeClassMethod(method, ...)
```

Polymorphic call of a class method

Remarks

cbject_Class must be defined before using this macro

Params

- method - Name of the method
- ... - Method params

Return

Depends on the called method

cbject_utils_invokeSuperMethod()

```
cbject_utils_invokeSuperMethod(type, method, ...)
```

Polymorphic call of a super method (object or class)

Remarks

cbject_Class must be defined before using this macro

Params

- klass - Name of the class
- method - Name of the method
- ...
    - object - cbject_Object reference (optional - in case of object method)
    - ... - Method params

Return

Depends on the called method

cbject_utils_Array_length()

```
cbject_utils_Array_length(array)
```

Gets length of an array

Params

- array - Array for which to get the length

cbject_utils_assertStatic()

```
cbject_utils_assertStatic(expression, identifier)
```

Compile time assert

Params

- expression - Expression to assert
- identifier - An identifier to describe the assertion

cbject_utils_Token_concat()

```
cbject_utils_Token_concat(token, otherToken)
```

Concatenates otherToken after the provided token

Params

- token - Token
- otherToken - Token to add after the provided token

cbject_utils_Token_concatIndirect()

```
cbject_utils_Token_concatIndirect(token, otherToken)
```

Concatenates otherToken after the provided token indirectly

Params

- token - Token
- otherToken - Token to add after the provided token

cbject_utils_Token_stringify()

```
cbject_utils_Token_stringify(token)
```

Stringifies the provided token

Params

- token - Token

cbject_utils_Token_stringifyIndirect()

```
cbject_utils_Token_stringifyIndirect(token)
```

Stringifies the provided token indirectly

Params

- token - Token

cbject_utils_VaArgs_getFirst()

```
cbject_utils_VaArgs_getFirst(...)
```

Gets first argument from *VA_ARGS*

Params

- ... - *VA_ARGS*

cbject_utils_VaArgs_getSecond()

```
cbject_utils_VaArgs_getSecond(...)
```

Gets second argument from *VA_ARGS*

Params

- ... - *VA_ARGS*

cbject_utils_VaArgs_getRest()

```
cbject_utils_VaArgs_getRest(...)
```

Gets list of arguments from *VA_ARGS* except the first

Remark

- Comma is added before the list
- Supports max 99 arguments

Params

- ... - *VA_ARGS*

cbject_utils_Pair_getFirst()

```
cbject_utils_Pair_getFirst(pair)
```

Gets first element from pair

Params

- pair - (first, second)

cbject_utils_Pair_getSecond()

```
cbject_utils_Pair_getSecond(pair)
```

Gets second element from pair

Params

- pair - (first, second)