

# Cbject docs

## Table of Contents

1. Overview .....	5
1.1. Features .....	5
1.2. Usage .....	5
1.3. cbject_Object model .....	5
2. API .....	8
2.1. cbject .....	8
2.1.1. Overview .....	8
2.2. cbject_config .....	8
2.2.1. Overview .....	8
2.2.2. Macros .....	8
cbject_config_useHeap .....	8
cbject_config_useStaticPool .....	8
cbject_config_useLinkedList .....	8
cbject_config_linkedListPoolSize .....	9
cbject_config_useNode .....	9
cbject_config_nodePoolSize .....	9
cbject_config_useSingleton .....	9
2.3. cbject_Object .....	10
2.3.1. Overview .....	10
2.3.2. Types .....	10
cbject_Object .....	10
cbject_ObjectClass .....	10
cbject_Object_PoolUsageStatus .....	10
cbject_Object_Source .....	11
struct cbject_Object .....	11
struct cbject_ObjectClass .....	12
2.3.3. Functions .....	13
cbject_ObjectClass_acquire() .....	13
cbject_ObjectClass_alloc() .....	13
cbject_Object_init() .....	13
cbject_Object_allocHelper() .....	13
cbject_Object_copy() .....	14
cbject_Object_equals() .....	14
cbject_Object_hashCode() .....	15
cbject_Object_retain() .....	15
cbject_Object_release() .....	15

cbject_Object_isOfType()	15
cbject_ObjectClass_instance()	16
2.3.4. Macros	16
cbject_ObjectClass_setup()	16
cbject_Object_class()	16
cbject_Object_instanceSize()	17
2.3.5. Tests	17
test_cbject_ObjectClass	17
test_cbject_Object_init	17
test_cbject_Object_equals	17
test_cbject_Object_hashCode	18
test_cbject_Object_isOfType	18
test_cbject_Object_copy	18
2.4. cbject_Singleton	19
2.4.1. Overview	19
2.4.2. Types	19
cbject_Singleton	19
cbject_SingletonClass	19
struct cbject_Singleton	20
struct cbject_SingletonClass	20
2.4.3. Functions	20
cbject_Singleton_init()	20
cbject_SingletonClass_instance()	20
2.5. cbject_Node	21
2.5.1. Overview	21
2.5.2. Types	21
cbject_Node	21
cbject_NodeClass	22
struct cbject_Node	22
struct cbject_NodeClass	22
2.5.3. Functions	23
cbject_Node_init()	23
cbject_Node_getElement()	23
cbject_Node_getPrevious()	23
cbject_Node_setPrevious()	24
cbject_Node_getNext()	24
cbject_Node_setNext()	24
cbject_NodeClass_instance()	24
2.5.4. Tests	25
test_cbject_Node_init	25
test_cbject_Node_setters	25

2.6. cbject_LinkedList	25
2.6.1. Overview	25
2.6.2. Types	26
cbject_LinkedList	26
cbject_LinkedListClass	26
cbject_LinkedList_NodeSource	27
struct cbject_LinkedList	27
struct cbject_LinkedListClass	27
2.6.3. Functions	28
cbject_LinkedList_init()	28
cbject_LinkedList_isEmpty()	28
cbject_LinkedList_addLast()	28
cbject_LinkedList_addFirst()	29
cbject_LinkedList_removeLast()	29
cbject_LinkedList_removeFirst()	29
cbject_LinkedList_clear()	29
cbject_LinkedList_getFirst()	30
cbject_LinkedList_getLast()	30
cbject_LinkedList_get()	30
cbject_LinkedList_getSize()	31
cbject_LinkedListClass_instance()	31
2.6.4. Tests	31
test_cbject_LinkedList_init	31
test_cbject_LinkedList_addFirst	31
test_cbject_LinkedList_addLast	32
test_cbject_LinkedList_removeFirst	32
test_cbject_LinkedList_removeLast	32
test_cbject_LinkedList_clear	32
2.7. cbject_utils	33
2.7.1. Overview	33
2.7.2. Macros	33
cbject_utils_acquire()	33
cbject_utils_alloc()	33
cbject_utils_stackAlloc()	34
cbject_utils_hashCode()	34
cbject_utils_equals()	34
cbject_utils_copy()	35
cbject_utils_retain()	35
cbject_utils_release()	35
cbject_utils_allocPool()	36
cbject_utils_nullPool	36

cbject_utils_doOnce .....	36
cbject_utils_invokeMethod() .....	37
cbject_utils_invokeClassMethod() .....	37
cbject_utils_invokeSuperMethod() .....	38
cbject_utils_Array_length() .....	38
cbject_utils_assertStatic() .....	38
cbject_utils_Token_concat() .....	39
cbject_utils_Token_concatIndirect() .....	39
cbject_utils_Token_stringify() .....	39
cbject_utils_Token_stringifyIndirect() .....	39
cbject_utils_VaArgs_getFirst() .....	40
cbject_utils_VaArgs_getSecond() .....	40
cbject_utils_VaArgs_getRest() .....	40
cbject_utils_Pair_getFirst() .....	40
cbject_utils_Pair_getSecond() .....	41

# 1. Overview

Cbobject 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 "cbobject.h"
```

*Example 2. How to create an object*

```
cbobject_Object * object =
cbobject_Object_init(cbobject_ObjectClass_alloc(cbobject_Object));
printf("%d\n", cbobject_Object_hashCode(object));
cbobject_utils_dealloc(object);
```

## 1.3. cbobject\_Object model

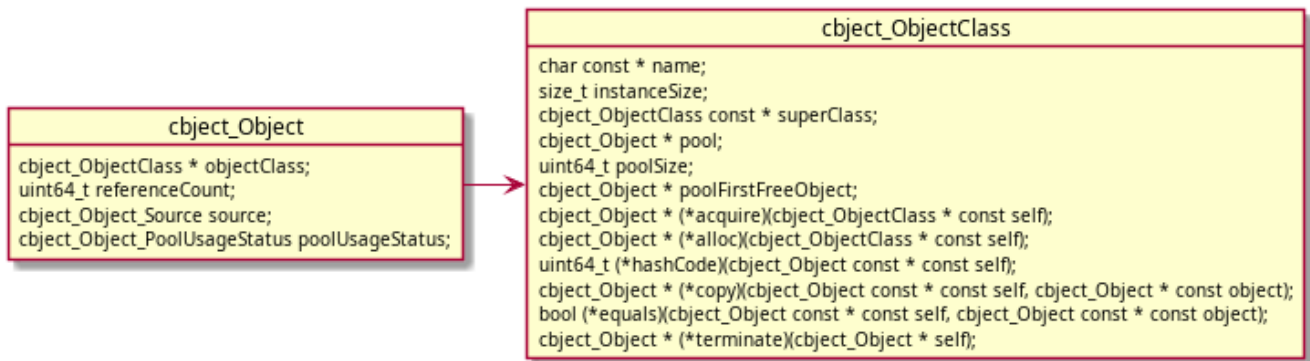


Figure 1. Building blocks

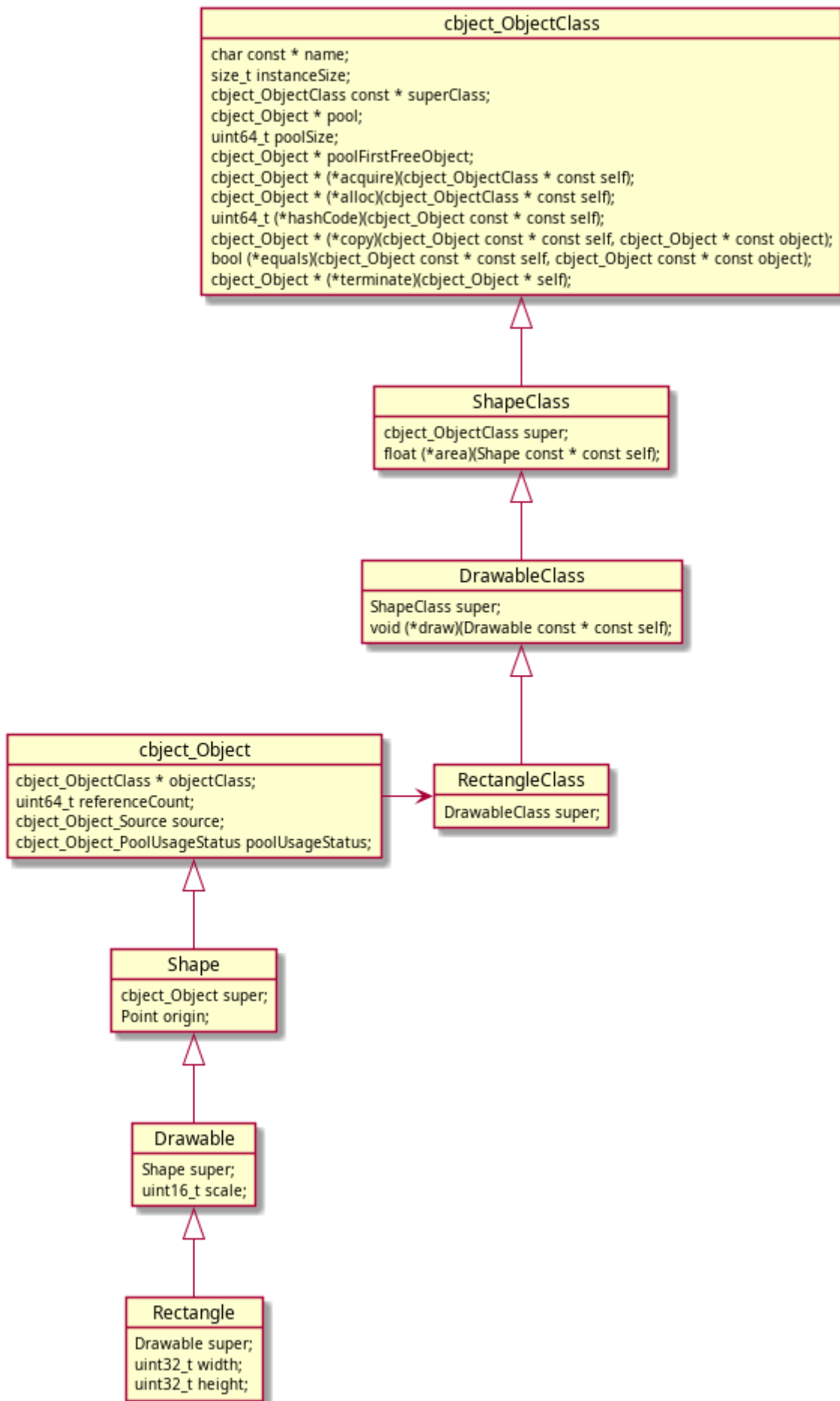


Figure 2. Rectangle class example

## 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

## object\_config\_linkedListPoolSize

```
#define object_config_linkedListPoolSize configValue
```

### LinkedList pool size config

#### Values

- $\geq 0$

## object\_config\_useNode

```
#define object_config_useNode configValue
```

### Node config

#### Values

- true
- false

## object\_config\_nodePoolSize

```
#define object_config_nodePoolSize configValue
```

### Node pool size config

#### Values

- $\geq 0$

## object\_config\_useSingleton

```
#define object_config_useSingleton configValue
```

Singleton config

Values

- true
- false

## 2.3. cbject\_Object

### 2.3.1. Overview

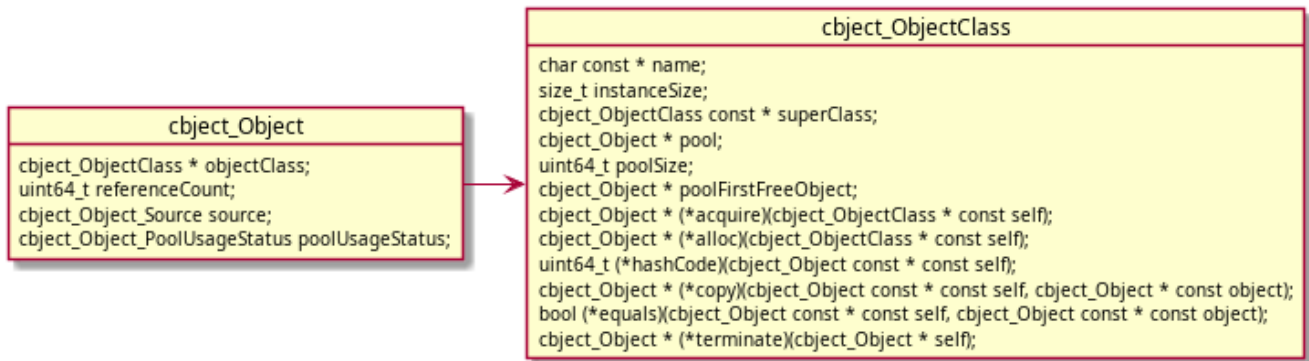


Figure 3. Context diagram

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 self);
    cbject_Object * (*alloc)(cbject_ObjectClass * const self);
    uint64_t (*hashCode)(cbject_Object const * const self);
    cbject_Object * (*copy)(cbject_Object const * const self, cbject_Object *
const object);
    bool (*equals)(cbject_Object const * const self, cbject_Object const * const
object);
    cbject_Object * (*terminate)(cbject_Object * self);
};
```

### 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\_ObjectClass\_acquire()**

```
cbject_Object * cbject_ObjectClass_acquire(cbject_ObjectClass * const self);
```

Acquires an object from the static pool

*Params*

- self - cbject\_ObjectClass reference

*Return*

Reference of the acquired object

#### **cbject\_ObjectClass\_alloc()**

```
cbject_Object * cbject_ObjectClass_alloc(cbject_ObjectClass * const self);
```

Allocates an object in heap memory

*Params*

- self - cbject\_ObjectClass reference

*Return*

Reference of the allocated object

#### **cbject\_Object\_init()**

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

Initializes an object

*Params*

- self - cbject\_Object reference

*Return*

Initialized object

#### **cbject\_Object\_allocHelper()**

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

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

*Params*

- self - cbject\_Object reference
- objectClass - cbject\_ObjectClass reference

*Return*

Reference to the object

### **cbject\_Object\_copy()**

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

Copies the object to the provided instance.

*Params*

- self - cbject\_Object reference
- object - Reference of a new object in which to copy the original one

*Return*

Reference of object

### **cbject\_Object\_equals()**

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

Compares two objects

*Params*

- self - cbject\_Object reference
- object - 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 self);
```

Gets the hash code of the object

### *Params*

- self - cbject\_Object reference

### *Return*

The hash code of the object

## **cbject\_Object\_retain()**

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

Increases the reference count of the object

### *Params*

- self - cbject\_Object reference

### *Return*

Reference to object

## **cbject\_Object\_release()**

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

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

### *Params*

- self - cbject\_Object reference

### *Return*

NULL

## **cbject\_Object\_isOfType()**

```
bool cbject_Object_isOfType(cbject_Object const * const self, cbject_ObjectClass
```

```
const * const objectClass);
```

Checks if an object is of a given class

*Params*

- self - 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(self)
```

Populates the class instance

*Remark*

cbject\_Class must be defined before using this macro

*Params*

- self - 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 and initialize it
2. Check if equals method returns true when comparing object to self
3. Allocate another object on stack and 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 and initialize it
2. Check if hashCode method returns the address in memory of the object

### **test\_cbject\_Object\_isOfType**

Test isOfType method

#### *Preconditions*

1. Define a dummy TestClass which extends cbject\_ObjectClass

#### *Steps*

1. Allocate object on stack and initialize it
2. Check if isOfType method returns true when checked against cbject\_Object
3. Check if isOfType method returns false when checked against Test

### **test\_cbject\_Object\_copy**

Test copy method

#### *Steps*

1. Allocate object on stack and 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

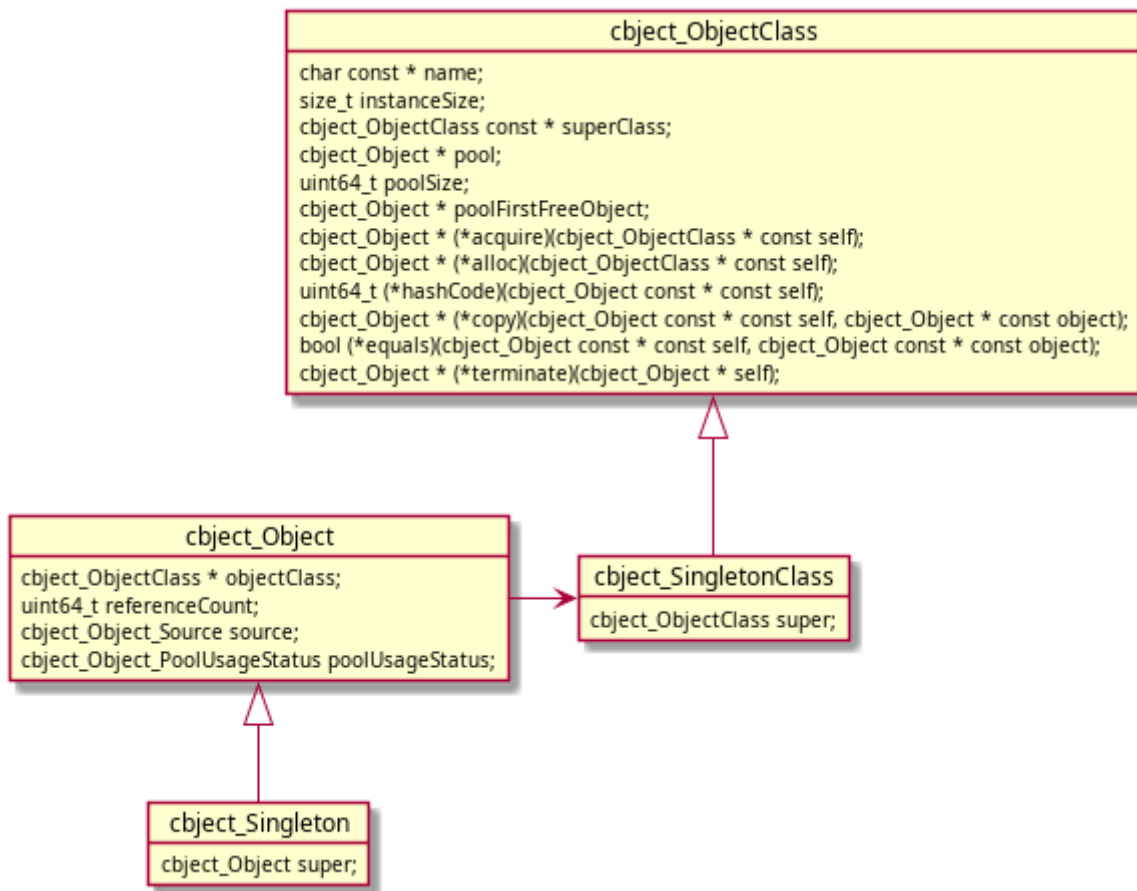


Figure 4. Context diagram

### 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 super;  
  
};
```

Definition of struct cbject\_Singleton

### *Members*

- super - Parent

## struct cbject\_SingletonClass

```
struct cbject_SingletonClass {  
    cbject_ObjectClass super;  
  
};
```

Definition of struct cbject\_SingletonClass

### *Members*

- super - Parent

## 2.4.3. Functions

### cbject\_Singleton\_init()

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

Initializes a singleton

### *Params*

- self - cbject\_Singleton reference

### *Return*

Initialized singleton

### cbject\_SingletonClass\_instance()

```
object_SingletonClass * object_SingletonClass_instance(void);
```

Gets object\_SingletonClass instance

*Return*

Reference of the class instance

## 2.5. cbject\_Node

### 2.5.1. Overview

Node data structure used in linked lists

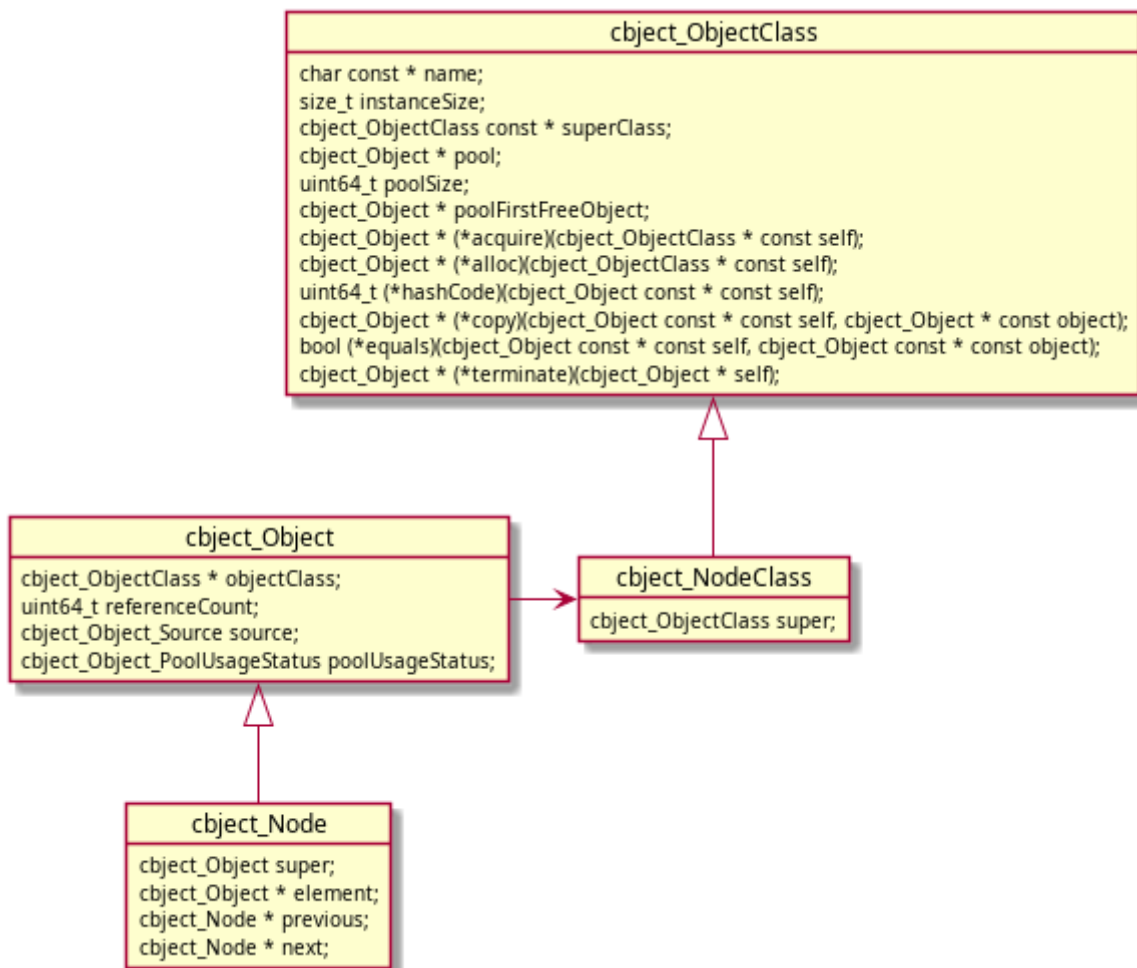


Figure 5. Context diagram

### 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 super;  
    cbject_Object * element;  
    cbject_Node * previous;  
    cbject_Node * next;  
  
};
```

Definition of struct cbject\_Node

#### *Members*

- super - 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*

- super - Parent

## 2.5.3. Functions

### **cbject\_Node\_init()**

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

Initializes a Node

#### *Params*

- self - 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 self);
```

Gets the data object contained in the node

#### *Params*

- self - cbject\_Node reference

#### *Return*

Data object in the node

### **cbject\_Node\_getPrevious()**

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

Gets the previous node

#### *Params*

- self - cbject\_Node reference

#### *Return*

The previous node

### **cbject\_Node\_setPrevious()**

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

Sets the previous node

*Params*

- self - cbject\_Node reference
- previousNode - cbject\_Node reference

### **cbject\_Node\_getNext()**

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

Gets the next node

*Params*

- self - cbject\_Node reference

*Return*

The next node

### **cbject\_Node\_setNext()**

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

Sets the next node

*Params*

- self - 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\_object\_Node\_init**

Test Node initialization

#### *Steps*

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

### **test\_object\_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. object\_LinkedList

### 2.6.1. Overview

Linked list data structure

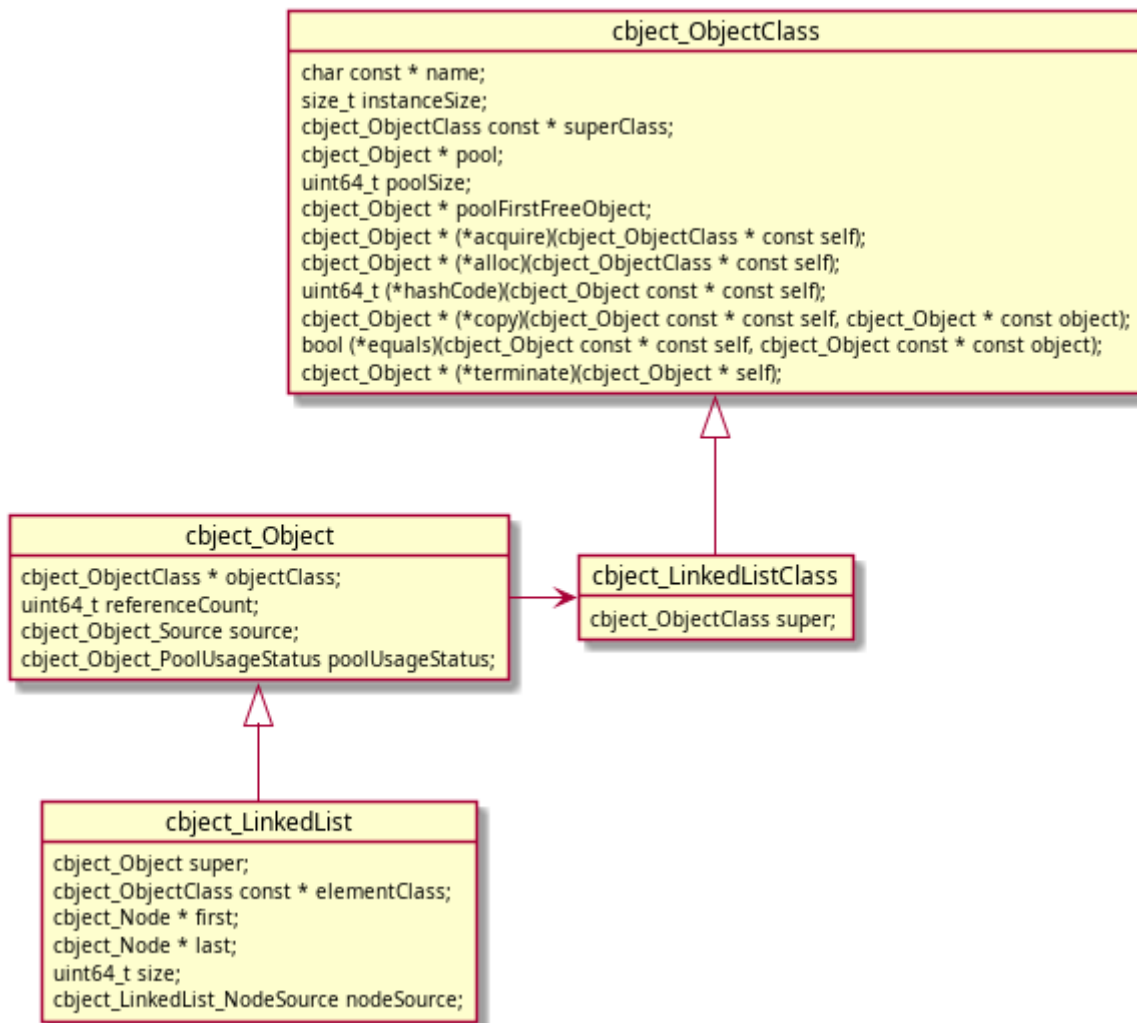


Figure 6. Context diagram

## 2.6.2. Types

### object\_LinkedList

```
typedef struct object_LinkedList object_LinkedList;
```

Typedef for struct `object_LinkedList`

### object\_LinkedListClass

```
typedef struct object_LinkedListClass object_LinkedListClass;
```

Typedef for struct `object_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 super;  
    cbject_ObjectClass const * elementClass;  
    cbject_Node * first;  
    cbject_Node * last;  
    uint64_t size;  
    cbject_LinkedList_NodeSource nodeSource;  
};
```

Definition of struct cbject\_LinkedList

### *Members*

- super - 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 super;
```

```
};
```

Definition of struct `object_LinkedListClass`

#### *Members*

- `super` - Parent

### 2.6.3. Functions

#### **`object_LinkedList_init()`**

```
object_LinkedList * object_LinkedList_init(  
    object_LinkedList * const self,  
    object_LinkedList_NodeSource const nodeSource  
);
```

Initializes a `LinkedList`

#### *Params*

- `self` - `object_LinkedList` reference
- `nodeSource` - Source for node creation (heap/staticPool) .Return Initialized and empty `LinkedList`

#### **`object_LinkedList_isEmpty()`**

```
bool object_LinkedList_isEmpty(object_LinkedList const * const self);
```

Checks if list is empty

#### *Params*

- `self` - `object_LinkedList` reference

#### *Return*

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

#### **`object_LinkedList_addLast()`**

```
void object_LinkedList_addLast(object_LinkedList * const self, object_Object *  
    const object);
```

Adds an element to the end of the list

*Params*

- self - cbject\_LinkedList reference
- object - Object to be added in the list

### **cbject\_LinkedList\_addFirst()**

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

Adds an element at the beginning of the list

*Params*

- self - cbject\_LinkedList reference
- object - Object to be added in the list

### **cbject\_LinkedList\_removeLast()**

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

Removes last element in the list

*Params*

- self - cbject\_LinkedList reference

### **cbject\_LinkedList\_removeFirst()**

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

Removes first element in the list

*Params*

- self - cbject\_LinkedList reference

### **cbject\_LinkedList\_clear()**

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

Removes all elements from the list

*Params*

- self - cbject\_LinkedList reference

### **cbject\_LinkedList\_getFirst()**

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

Gets the first element in the list

*Params*

- self - cbject\_LinkedList reference

*Return*

First element in list

### **cbject\_LinkedList\_getLast()**

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

Gets the last element in the list

*Params*

- self - cbject\_LinkedList reference

*Return*

Last element in list

### **cbject\_LinkedList\_get()**

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

Gets element at specified index

*Params*

- self - 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 self);
```

Gets the size of the list (number of elements)

#### *Params*

- self - 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\_object\_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\_object\_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\_object\_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\_object\_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(type)
```

Acquires an object from the static pool

##### *Remarks*

Calls `cbject_ObjectClass_acquire()` and does the necessary casting

##### *Params*

- type - Name of class

##### *Return*

Reference of the acquired object

#### **cbject\_utils\_alloc()**

```
cbject_utils_alloc(type)
```

Allocates an object in heap memory

##### *Remarks*

Calls `cbject_ObjectClass_alloc()` and does the necessary casting

##### *Params*

- type - Name of class

##### *Return*

Reference of the allocated object

## **cbject\_utils\_stackAlloc()**

```
cbject_utils_stackAlloc(type)
```

Allocates an object on the stack

### *Params*

- type - Name of class

### *Return*

Reference of the allocated memory

## **cbject\_utils\_hashCode()**

```
cbject_utils_hashCode(self)
```

Gets the hash code of the object

### *Remarks*

Calls cbject\_Object\_hashCode() and does the necessary casting

### *Params*

- self - cbject\_Object reference

### *Return*

The hash code of the object

## **cbject\_utils\_equals()**

```
cbject_utils_equals(self, object)
```

Compares two objects

### *Remarks*

Calls cbject\_Object\_equals() and does the necessary casting

### *Params*

- self - cbject\_Object reference
- object - Reference for the compared object

### *Return*

- true - If the objects are equal

- false - If the objects are different

### **object\_utils\_copy()**

```
object_utils_copy(self, object)
```

Copies the object to the provided instance.

#### *Remarks*

Calls `object_Object_copy()` and does the necessary casting

#### *Params*

- self - `object_Object` reference
- object - Reference of a new object in which to copy the original one

#### *Return*

Reference of object

### **object\_utils\_retain()**

```
object_utils_retain(self)
```

Increases the reference count of the object

#### *Remarks*

Calls `object_Object_retain()` and does the necessary casting

#### *Params*

- self - `object_Object` reference

#### *Return*

Reference to object

### **object\_utils\_release()**

```
object_utils_release(self)
```

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*

- `self` - `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\_nullPool**

```
cbject_utils_nullPool
```

Declares a null static pool

*Remarks*

`cbject_Class` must be defined before using this macro Use instead of `cbject_utils_allocPool` if no static pool is needed

## **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*

- type - Name of the class
- method - Name of the method
- ...
  - self - 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(self)
```

Gets length of an array

### *Params*

- self - 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

### **object\_utils-Token\_concat()**

```
object_utils-Token_concat(self, token)
```

Concatenates otherToken after the provided token

#### *Params*

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

### **object\_utils-Token\_concatIndirect()**

```
object_utils-Token_concatIndirect(self, token)
```

Concatenates otherToken after the provided token indirectly

#### *Params*

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

### **object\_utils-Token\_stringify()**

```
object_utils-Token_stringify(self)
```

Stringifies the provided token

#### *Params*

- self - Token

### **object\_utils-Token\_stringifyIndirect()**

```
object_utils-Token_stringifyIndirect(self)
```

Stringifies the provided token indirectly

#### *Params*

- self - 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(self)
```

Gets first element from pair

*Params*

- self - (first, second)



## **object\_utils\_Pair\_getSecond()**

```
object_utils_Pair_getSecond(self)
```

Gets second element from pair

### *Params*

- self - (first, second)