

# Cbject docs

## Table of Contents

1. Overview .....	3
1.1. Features .....	3
1.2. Usage .....	3
1.3. cbject_Object model .....	3
2. API .....	4
2.1. cbject_Object .....	5
2.1.1. Overview .....	5
2.1.2. Types .....	5
cbject_Object .....	5
cbject_ObjectClass .....	5
struct cbject_Object .....	5
struct cbject_ObjectClass .....	5
2.1.3. Functions .....	6
cbject_Object_alloc() .....	6
cbject_Object_init() .....	6
cbject_Object_copy() .....	7
cbject_Object_equals() .....	7
cbject_Object_hashCode() .....	7
cbject_Object_terminate() .....	8
cbject_Object_dealloc() .....	8
cbject_Object_isOfClass() .....	8
cbject_ObjectClass_instance() .....	9
2.1.4. Macros .....	9
cbject_Class_setup() .....	9
cbject_Object_class() .....	9
cbject_Object_instanceSize() .....	10
2.1.5. Tests .....	10
test_cbject_ObjectClass .....	10
test_cbject_Object_init .....	10
test_cbject_Object_equals .....	10
test_cbject_Object_hashCode .....	11
test_cbject_Object_isOfClass .....	11
test_cbject_Object_copy .....	11
2.2. cbject_Singleton .....	11
2.2.1. Overview .....	11
2.2.2. Types .....	11

cbject_Singleton .....	12
cbject_SingletonClass .....	12
struct cbject_Singleton .....	12
struct cbject_SingletonClass .....	12
2.2.3. Functions .....	12
cbject_Singleton_init() .....	13
cbject_SingletonClass_instance() .....	13
2.3. cbject_utils .....	13
2.3.1. Overview .....	13
2.3.2. Macros .....	13
cbject_utils-Token_concat() .....	13
cbject_utils-Token_concatIndirect() .....	14
cbject_utils-Token_stringify() .....	14
cbject_utils-Token_stringifyIndirect() .....	14
cbject_utils_VaArgs_getFirst() .....	14
cbject_utils_VaArgs_getSecond() .....	15
cbject_utils_VaArgs_getRest() .....	15
cbject_utils_Pair_getFirst() .....	15
cbject_utils_Pair_getSecond() .....	15
2.4. cbject .....	16
2.4.1. Overview .....	16
2.4.2. Macros .....	16
cbject_alloc() .....	16
cbject_salloc() .....	16
cbject_hashCode() .....	16
cbject_equals() .....	17
cbject_copy() .....	17
cbject_terminate() .....	17
cbject_dealloc() .....	18
cbject_Array_length() .....	18
cbject_assertStatic() .....	18
cbject_doOnce .....	18
cbject_invokeMethod() .....	19
cbject_invokeClassMethod() .....	19
cbject_invokeSuperMethod() .....	20

# 1. Overview

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

## 1.1. Features

- Objects
- Classes
- Inheritance
- Polymorphism

## 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_alloc(cbobject_Object));
printf("%d\n", cbobject_Object_hashCode(object));
cbobject_dealloc(object);
```

## 1.3. cbobject\_Object model

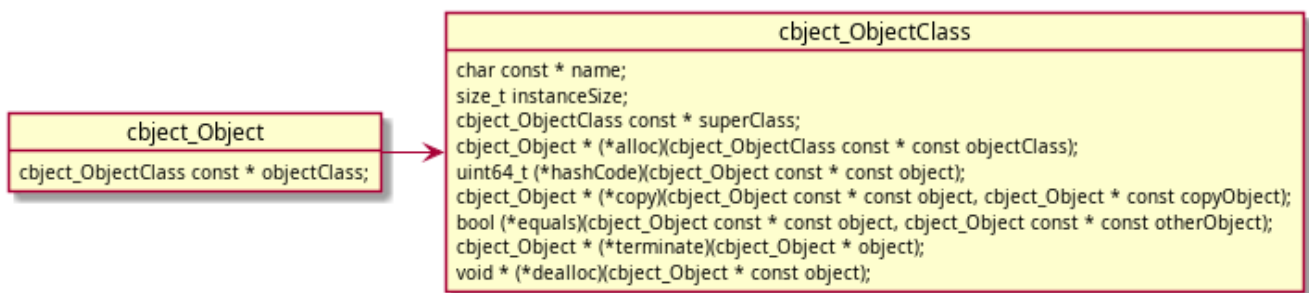


Figure 1. Building blocks

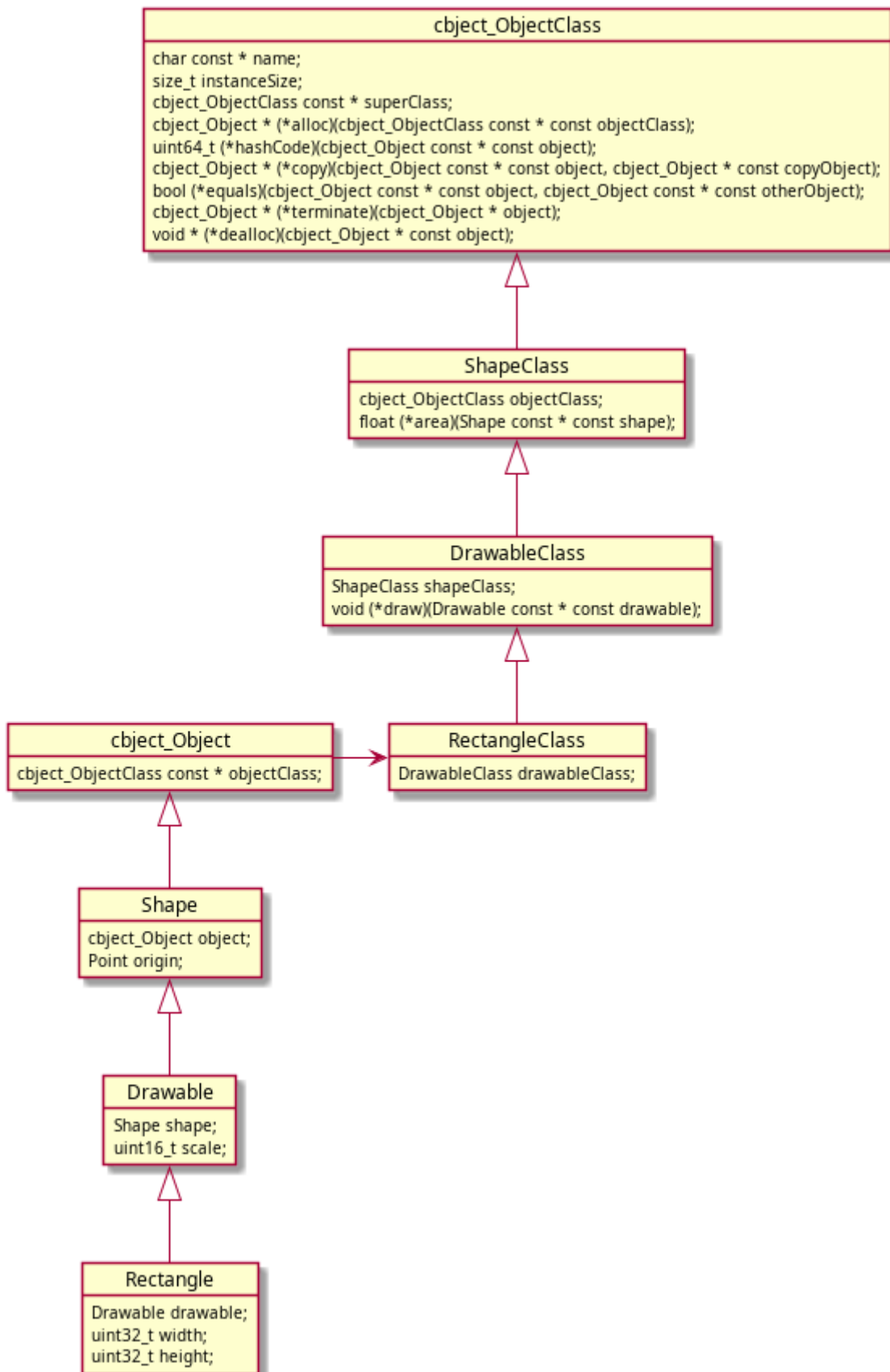


Figure 2. Rectangle class example

## 2. API

## 2.1. cbject\_Object

### 2.1.1. Overview

The building block. All objects defined in Cbject need to extend cbject\_Object.

### 2.1.2. 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

#### struct cbject\_Object

```
struct cbject_Object {  
    cbject_ObjectClass const * objectClass;  
};
```

Definition of struct cbject\_Object

##### *Members*

- objectClass - cbject\_ObjectClass reference

#### struct cbject\_ObjectClass

```
struct cbject_ObjectClass {  
    char const * name;  
    size_t instanceSize;  
    cbject_ObjectClass const * superClass;  
    cbject_Object * (*alloc)(cbject_ObjectClass const * 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);
    void * (*dealloc)(cbject_Object * const 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
- alloc - Alloc method reference
- hashCode - Hash code method reference
- copy - Copy method reference
- equals - Equals method reference
- terminate - Terminate method reference
- dealloc - Dealloc method reference

## 2.1.3. Functions

### **cbject\_Object\_alloc()**

```
cbject_Object * cbject_Object_alloc(cbject_ObjectClass const * 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\_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 allocated 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\_terminate()**

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

Terminates an object.

#### *Params*

- object - cbject\_Object reference

#### *Return*

NULL

### **cbject\_Object\_dealloc()**

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

Deallocates memory for an object

#### *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 const * cbject_ObjectClass_instance(void);
```

Gets cbject\_ObjectClass instance

*Return*

Reference of the class instance

## **2.1.4. Macros**

### **cbject\_Class\_setup()**

```
cbject_Class_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

## **object\_Object\_instanceSize()**

```
object_Object_instanceSize(object)
```

Gets the size in memory of an object

### *Params*

- object - object\_Object reference

### *Return*

The size in memory of the object

## **2.1.5. Tests**

### **test\_object\_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\_object\_Object\_init**

Test initialization of object\_Object

### *Steps*

1. Allocate object on stack and initialize it
2. Check if object class points to object\_ObjectClass instance

### **test\_object\_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 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

## **2.2. cbject\_Singleton**

### **2.2.1. Overview**

The building block. All objects defined in Cbject need to extend cbject\_Singleton.

### **2.2.2. 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

## **2.2.3. 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 const * cbject_SingletonClass_instance(void);
```

Gets cbject\_SingletonClass instance

### *Return*

Reference of the class instance

## **2.3. cbject\_utils**

### **2.3.1. Overview**

TODO

### **2.3.2. Macros**

## **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*

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

```
object_utils_VaArgs_getSecond(...)
```

Gets second argument from *VA\_ARGS*

*Params*

- ... - *VA\_ARGS*

### **object\_utils\_VaArgs\_getRest()**

```
object_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*

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

```
object_utils_Pair_getFirst(pair)
```

Gets first element from pair

*Params*

- pair - (first, second)

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

```
object_utils_Pair_getSecond(pair)
```

Gets second element from pair

*Params*

- pair - (first, second)

## 2.4. cbject

### 2.4.1. Overview

todo

### 2.4.2. Macros

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

```
cbject_alloc(klass)
```

Syntactic sugar to allocate an object in heap memory

*Params*

- klass - Name of class

*Return*

Reference of the allocated object

#### **cbject\_salloc()**

```
cbject_salloc(klass)
```

Syntactic sugar to allocate object on the stack

*Params*

- klass - Name of class

*Return*

Reference of the allocated memory

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

```
cbject_hashCode(object)
```

Syntactic sugar to get the hash code of the object

*Params*

- object - cbject\_Object reference

*Return*



The hash code of the object

## **object\_equals()**

```
object_equals(object, otherObject)
```

Syntactic sugar to compare two objects

### *Params*

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

### *Return*

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

## **object\_copy()**

```
object_copy(object, copyObject)
```

Syntactic sugar to copy the object to the provided instance.

### *Params*

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

### *Return*

Reference of copyObject

## **object\_terminate()**

```
object_terminate(object)
```

Syntactic sugar to terminate an object.

### *Params*

- object - object reference

### *Return*

NULL

### **cbject\_dealloc()**

```
cbject_dealloc(object)
```

Syntactic sugar to free memory allocated for an object

#### *Params*

- object - cbject\_Object reference

#### *Return*

NULL

### **cbject\_Array\_length()**

```
cbject_Array_length(array)
```

Gets length of an array

#### *Params*

- array - Array for which to get the length

### **cbject\_assertStatic()**

```
cbject_assertStatic(expression, identifier)
```

Compile time assert

#### *Params*

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

### **cbject\_doOnce**

```
cbject_doOnce
```

Runs a block of code only once

#### *Usage*

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

#### *Remark*

Not thread safe

### **cbject\_invokeMethod()**

```
cbject_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\_invokeClassMethod()**

```
cbject_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\_invokeSuperMethod()**

```
cbject_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