

# Cbject docs

## Table of Contents

1. Overview .....	4
1.1. Features .....	4
1.2. Usage .....	4
1.3. Object model .....	4
2. API .....	5
2.1. Object .....	5
2.1.1. Overview .....	5
2.1.2. Types .....	5
Object_Class .....	5
Object .....	6
struct Object_Class .....	6
struct Object .....	6
2.1.3. Functions .....	7
Object_Class_0 .....	7
Object_alloc() .....	7
Object_dealloc() .....	7
Object_init() .....	8
Object_tearardown() .....	8
Object_copy() .....	8
Object_equals() .....	9
Object_hashCode() .....	9
Object_isOfClass() .....	9
2.1.4. Macros .....	10
class_0 .....	10
initClass_0 .....	10
setUpClass_0 .....	10
overrideClassMethod_0 .....	11
initObject_0 .....	11
sallocInit_0 .....	11
classOf_0 .....	12
setUpObject_0 .....	12
objectSizeOf_0 .....	13
traitOf_0 .....	13
objectMethodCall_0 .....	13
classMethodCall_0 .....	14
alloc_0 .....	14

allocInit_0 .....	15
dealloc_0 .....	15
teardown_0 .....	15
copy_0 .....	16
allocCopy_0 .....	16
sallocCopy_0 .....	16
equals_0 .....	17
hashCode_0 .....	17
isOfClass_0 .....	17
2.1.5. Tests .....	18
test_Object_class .....	18
test_Object_init .....	18
test_Object_equals .....	18
test_Object_hashCode .....	19
test_Object_isOfClass .....	19
test_Object_copy .....	19
2.2. Trait .....	20
2.2.1. Overview .....	20
2.2.2. Types .....	20
Trait_Interface .....	20
Trait .....	20
struct Trait_Interface .....	20
struct Trait .....	21
2.2.3. Functions .....	21
Trait_Interface_ .....	21
Trait_init .....	21
2.2.4. Macros .....	22
interface_0 .....	22
initInterface_0 .....	22
setUpInterface_0 .....	22
overrideInterfaceMethod_0 .....	23
offsetOf_0 .....	23
objectOf_0 .....	23
interfaceOffsetOf_0 .....	24
interfaceOf_0 .....	24
initTrait_0 .....	24
setUpTrait_0 .....	25
traitMethodCall_0 .....	25
interfaceMethodCall_0 .....	26
2.3. DoOnce .....	26
2.3.1. Overview .....	26

2.3.2. Macros .....	27
doOnce_0 .....	27

# 1. Overview

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

## 1.1. Features

- Classes
- Objects
- Traits
- Interfaces
- Inheritance
- Polymorphism

## 1.2. Usage

*Example 1. How to add it to a project*

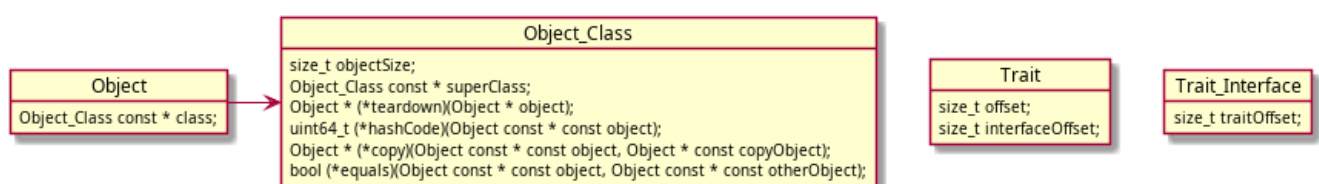
Include the following header file:

```
#include "Object.h"
```

*Example 2. How to create an object*

```
Object * object = allocInit_(Object);
printf("%d\n", hashCode_(object));
dealloc_(object);
```

## 1.3. Object model



*Figure 1. Building blocks*

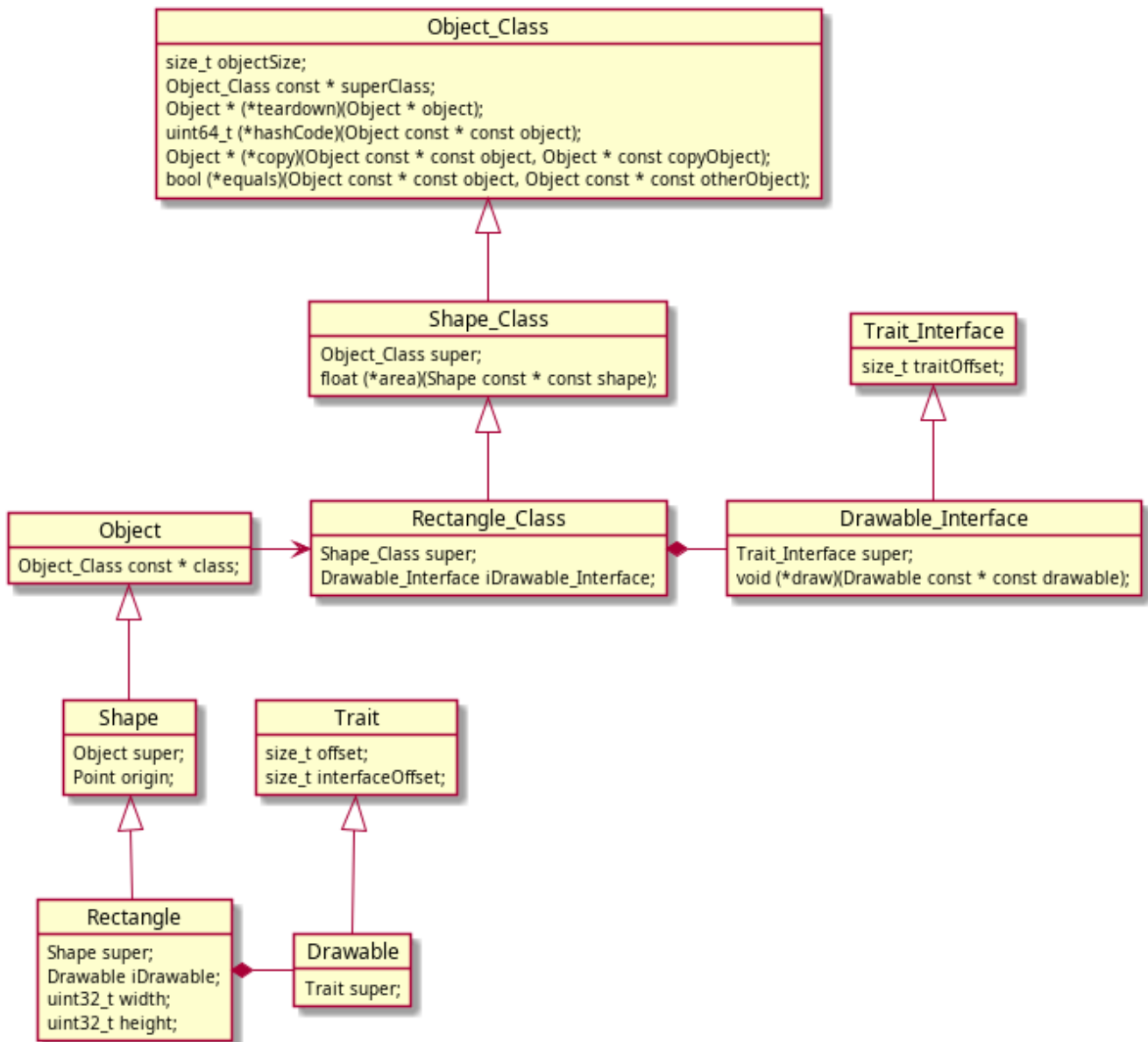


Figure 2. Rectangle class example

## 2. API

### 2.1. Object

#### 2.1.1. Overview

The building block. All objects defined in Cbjet need to extend Object.

#### 2.1.2. Types

**Object\_Class**

```
typedef struct Object_Class Object_Class;
```

Typedef for struct Object\_Class

## Object

```
typedef struct Object Object;
```

Typedef for struct Object

## struct Object\_Class

```
struct Object_Class {  
    size_t objectSize;  
    Object_Class const * superClass;  
    Object * (*teardown)(Object * object);  
    uint64_t (*hashCode)(Object const * const object);  
    Object * (*copy)(Object const * const object, Object * const copyObject);  
    bool (*equals)(Object const * const object, Object const * const otherObject);  
};
```

Definition of struct Object\_Class

### *Members*

- objectSize - Size in memory of object
- superClass - Super class of object
- teardown - Function pointer for the teardown method
- hashCode - Function pointer for the hash code method
- copy - Function pointer for the copy method
- equals - Function pointer for the equals method

## struct Object

```
struct Object {
    Object_Class const * class;
};
```

Definition of struct Object

*Members*

- class - Pointer to the class structure

## 2.1.3. Functions

### Object\_Class\_O

```
Object_Class const * Object_Class_(void);
```

Get Object\_Class instance

*Return*

Reference of the class instance

### Object\_alloc()

```
Object * Object_alloc(Object_Class const * const class);
```

Allocate an object in heap memory

*Params*

- class - Class reference

*Return*

Reference of the allocated object

### Object\_dealloc()

```
Object * Object_dealloc(Object * const object);
```

Free memory allocated for an object

*Params*

- object - Object reference

*Return*

Always returns NULL

## **Object\_init()**

```
Object * Object_init(Object * const object);
```

Initialize an object

*Params*

- object - Object reference

*Return*

Initialized object

## **Object\_tearardown()**

```
Object * Object_tearardown(Object * object);
```

Tearardown an object.

*Params*

- object - Object reference

*Return*

Always returns NULL

## **Object\_copy()**



```
Object * Object_copy(Object const * const object, Object * const copyObject);
```

Make a copy of an object.

*Params*

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

*Return*

Pointer to a new object (copy of the original one)

## **Object\_equals()**

```
bool Object_equals(Object const * const object, Object const * const otherObject);
```

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\_hashCode()**

```
uint64_t Object_hashCode(Object const * const object);
```

Get hash code of object

*Params*

- object - Object reference

*Return*

Object hash code

## **Object\_isOfClass()**

```
bool Object_isOfClass(Object const * const object, Object_Class const * const class);
```

Check if an object is of a given class

*Params*

- object - Object reference
- class - Class reference

*Return*

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

## 2.1.4. Macros

### **class\_()**

```
#define class_(className)
```

Syntactic sugar to get class reference

*Params*

- className - Name of the class

*Return*

Class reference

### **initClass\_()**

```
#define initClass_(className, object)
```

Initialize a class

*Params*

- className - Name of the class
- object - Class reference

### **setUpClass\_()**

```
#define setUpClass_(className, superClassName, class)
```

Class setup (initialize, set the object size and super class)

*Params*

- className - Name of the class
- superClassName - Name of the super class
- class - Class reference

## **overrideClassMethod\_()**

```
#define overrideClassMethod_(className, class, methodName)
```

Override a method of a class

*Params*

- className - Name of the class
- class - Class reference
- methodName - Name of the method

## **initObject\_()**

```
#define initObject_(className, ...)
```

Syntactic sugar for object initialization

*Params*

- className - Name of the class
- ...
  - object - Object reference
  - ... - Init params

*Return*

Initialized object

## **sallocInit\_()**

```
#define sallocInit(...)
```

Syntactic sugar to allocate and init an object in stack memory

*Params*

- ...
  - className - Name of class
  - ... - Init params

*Return*

Reference of the allocated and initialized object

## **classOf\_()**

```
#define classOf_(object)
```

Get the class of an object

*Params*

- object - Object reference

*Return*

Class reference

## **setUpObject\_()**

```
#define setUpObject_(className, superClassName, ...)
```

Object setup (initialize, set the object class)

*Params*

- className - Name of the class
- superClassName - Name of the super class
- ...
  - object - Object reference
  - ... - Init params

## **objectSizeOf\_()**

```
#define objectSizeOf_(object)
```

Get the size in memory of an object

### *Params*

- object - Object reference

### *Return*

Object size

## **traitOf\_()**

```
#define traitOf_(object, className, interfaceName)
```

Get trait of an object

### *Params*

- object - Object reference
- className - Name of the class
- interfaceName - Name of the interface

### *Return*

Trait reference

## **objectMethodCall\_()**

```
#define objectMethodCall_(className, methodName, ...)
```

Call a method through an object

*Params*

- className - Name of the class
- methodName - Name of the method
- ...
  - object - Object reference
  - ... - Method params

*Return*

Depends on the called method

## **classMethodCall\_()**

```
#define classMethodCall_(className, superClassName, methodName, ...)
```

Call a method through a class

*Params*

- className - Name of the class
- superClassName - Name of the super class
- methodName - Name of the method
- ...
  - object - Object reference
  - ... - Method params

*Return*

Depends on the called method

## **alloc\_()**

```
#define alloc_(className)
```

Syntactic sugar to allocate an object in heap memory

*Params*

- className - Name of class

*Return*

Reference of the allocated object

## **allocInit\_()**

```
#define allocInit_(...)
```

Syntactic sugar to allocate and init an object in heap memory

*Params*

- ...
  - className - Name of class
  - ... - Init params

*Return*

Reference of the allocated and initialized object

## **dealloc\_()**

```
#define dealloc_(object)
```

Syntactic sugar to free memory allocated for an object

*Params*

- object - Object reference

*Return*

Always returns NULL

## **teardown\_()**

```
#define teardown_(object)
```

Syntactic sugar to teardown an object.

*Params*

- object - Object reference

*Return*

Always returns NULL

## **copy\_0**

```
#define copy_(className, object, copyObject)
```

Syntactic sugar to make a copy of an object.

*Params*

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

*Return*

Pointer to a new object (copy of the original one)

## **allocCopy\_0**

```
#define allocCopy_(className, object)
```

Syntactic sugar to copy object in new object allocated in heap memory

*Params*

- className - Name of class
- object - Object reference

*Return*

Reference of the allocated object (copy of the original one)

## **sallocCopy\_0**



```
#define sallocCopy(className, object)
```

Syntactic sugar to copy object in new object allocated in stack memory

*Params*

- className - Name of class
- object - Object reference

*Return*

Reference of the allocated object (copy of the original one)

## **equals\_()**

```
#define 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

## **hashCode\_()**

```
#define hashCode_(object)
```

Syntactic sugar to get hash code of object

*Params*

- object - Object reference

*Return*

Object hash code

## **isOfClass\_()**

```
#define isOfClass_(object, className)
```

Syntactic sugar to check if an object is of a given class

*Params*

- object - Object reference
- className - Class name

*Return*

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

## 2.1.5. Tests

### test\_Object\_class

Test setup of Object\_Class

*Steps*

1. Get Object\_Class 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\_init

Test initialization of Object

*Steps*

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

### test\_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\_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\_Object\_isOfClass**

Test isOfClass method

*Preconditions*

1. Define a dummy Test\_Class which extends Object\_Class

*Steps*

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

### **test\_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

## 2.2. Trait

### 2.2.1. Overview

TODO

### 2.2.2. Types

**Trait\_Interface**

```
typedef struct Trait_Interface Trait_Interface;
```

Typedef for struct Trait\_Interface

**Trait**

```
typedef struct Trait Trait;
```

Typedef for struct Trait

**struct Trait\_Interface**

```
struct Trait_Interface {
    size_t traitOffset;
};
```

Definition of struct Trait\_Interface

*Members*

- traitOffset - Offset of trait in containing object

## struct Trait

```
struct Trait {
    size_t offset;
    size_t interfaceOffset;
};
```

Definition of struct Trait

*Members*

- offset - Offset of Trait in container Object
- interfaceOffset - Offset of Trait\_Interface in container Object\_Class

## 2.2.3. Functions

### Trait\_Interface\_

```
Trait_Interface const * Trait_Interface_(void);
```

Get Trait\_Interface instance

*Return*

Reference of the trait interface

### Trait\_init

```
Trait * Trait_init(Trait * const trait);
```

Initialize a trait

*Params*

- trait - Trait reference

*Return*

Initialized trait

## 2.2.4. Macros

### interface\_()

```
#define interface_(interfaceName)
```

Syntactic sugar to get interface reference

*Params*

- interfaceName - Name of the interface

*Return*

Interface reference

### initInterface\_()

```
#define initInterface_(interfaceName, interface)
```

Initialize an interface

*Params*

- interfaceName - Name of the interface
- interface - Interface reference

### setUpInterface\_()

```
#define setUpInterface_(className, interfaceName, interface)
```

Interface setup (initialize, set the trait offset in container object)

*Params*

- className - Name of the class
- interfaceName - Name of the interface
- interface - Interface reference

## **overrideInterfaceMethod\_()**

```
#define overrideInterfaceMethod_(className, interfaceName, interface, methodName)
```

Override a method of an interface

*Params*

- className - Name of the class
- interfaceName - Name of the interface
- interface - Interface reference
- methodName - Name of the method

## **offsetOf\_()**

```
#define offsetOf_(trait)
```

Get offset of a trait in container object

*Params*

- trait - Trait reference

*Return*

Offset of trait in container object

## **objectOf\_()**

```
#define objectOf_(trait)
```

Get container object from a trait

*Params*

- trait - Trait reference

*Return*

Reference of the container object

## **interfaceOffsetOf\_()**

```
#define interfaceOffsetOf_(trait)
```

Get the interface offset in container class

*Params*

- trait - Trait reference

*Return*

Offset of interface in container class

## **interfaceOf\_()**

```
#define interfaceOf_(trait)
```

Get the interface of a trait

*Params*

- trait - Trait reference

*Return*

Interface reference

## **initTrait\_()**



```
#define initTrait_(interfaceName, ...)
```

Syntactic sugar for trait initialization

*Params*

- interfaceName - Name of the interface
- ...
  - trait - Trait reference
  - ... - Init params

*Return*

Initialized trait

## **setUpTrait\_()**

```
#define setUpTrait_(className, interfaceName, ...)
```

Trait setup (initialize, set the trait offset and interface offset)

*Params*

- className - Name of the class
- interfaceName - Name of the interface
- ...
  - trait - Trait reference
  - ... - Init params

## **traitMethodCall\_()**

```
#define traitMethodCall_(interfaceName, methodName, ...)
```

Call a method through a trait

*Params*

- interfaceName - Name of the interface
- methodName - Name of the method
- ...
  - trait - Trait reference
  - ... - Method params

*Return*

Depends on the called method

## **interfaceMethodCall\_()**

```
#define interfaceMethodCall_(className, interfaceName, methodName, ...)
```

Call a method through an interface

*Params*

- className - Name of the class
- interfaceName - Name of the interface
- methodName - Name of the method
- ...
  - trait - Trait reference
  - ... - Method params

*Return*

Depends on the called method

## **2.3. DoOnce**

### **2.3.1. Overview**

Implementation for doOnce\_() macro

## 2.3.2. Macros

### doOnce\_()

```
#define doOnce_(block)
```

Run a block of code only once

#### *Remark*

It uses a mutex to provide thread safety. The mutex settings need to be changed based on the concurrency framework used. By default pthread is used.

#### *Usage example*

```
doOnce_({  
    functionCall();  
});
```

#### *Params*

- block - The block of code to be run