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

1.2. Usage

Example 1. How to add it to a project

Include the following header file:

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

Example 2. How to create an object

```c
cbject_Object * object = cbject_Object_init(cbject_alloc(cbject_Object));
printf("%d\n", cbject_Object_hashCode(object));
cbject_dealloc(object);
```

1.3. cbject_Object model

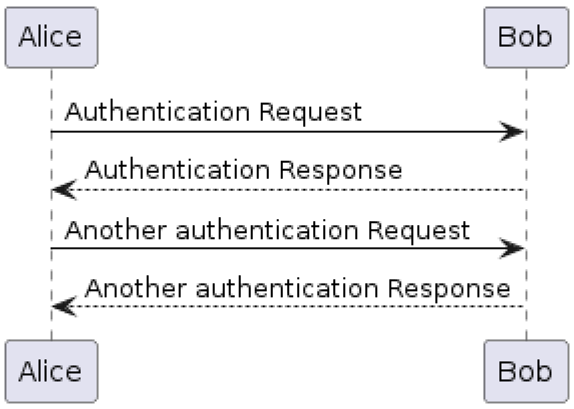
Alice
Bob
Authentication Request
Authentication Response
Another authentication Request
Another authentication Response
Alice
Bob

Figure 1. Building blocks

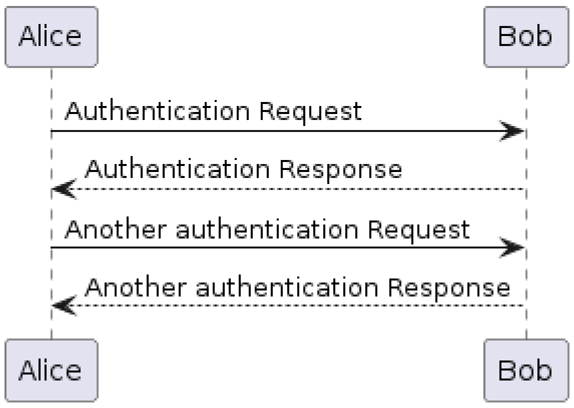
Alice
Bob
Authentication Request
Authentication Response
Another authentication Request
Another authentication Response
Alice
Bob

Figure 2. Building blocks

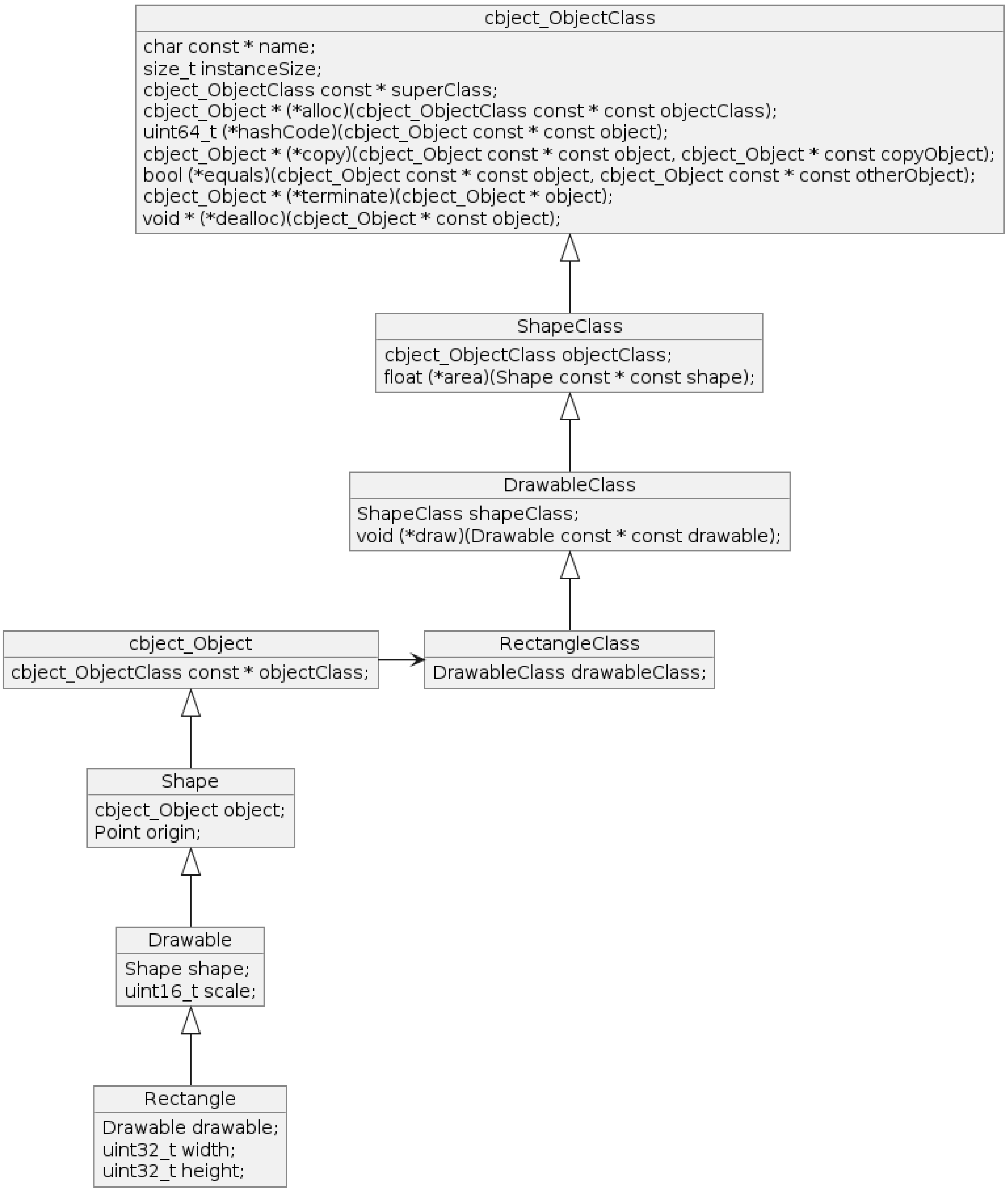
cbject_ObjectClass
char const * name;
size_t instanceSize;
cbject_ObjectClass const * superClass;
cbject_Object * (*alloc)(cbject_ObjectClass const * const objectClass);
uint64_t (*hashCode)(cbject_Object const * const object);
cbject_Object * (*copy)(cbject_Object const * const object, cbject_Object * 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);
ShapeClass
cbject_ObjectClass objectClass;
float (*area)(Shape const * const shape);
DrawableClass
ShapeClass shapeClass;
void (*draw)(Drawable const * const drawable);
cbject_Object
cbject_ObjectClass const * objectClass;
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 3. 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

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

2.1.5. 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

2.2. 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*

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)

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

cbject_equals()

```
cbject_equals(object, otherObject)
```

Syntactic sugar to compare 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_copy()

```
cbject_copy(object, copyObject)
```

Syntactic sugar to copy 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_terminate()

```
cbject_terminate(object)
```

Syntactic sugar to terminate an object.

Params

- object - cbject_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