Cbject docs

Table of Contents

1. Overview

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

1.1. Features

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

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_allocInit(Cbject_Object);
printf("%d\n", Cbject_hashCode(object));
Cbject_dealloc(object);
```

1.3. Cbject_Object model

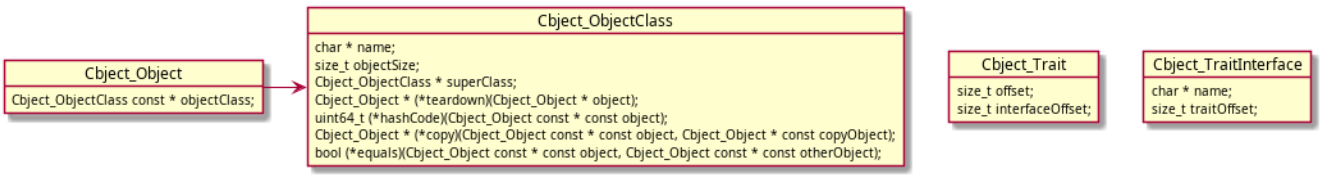

Figure 1. Building blocks

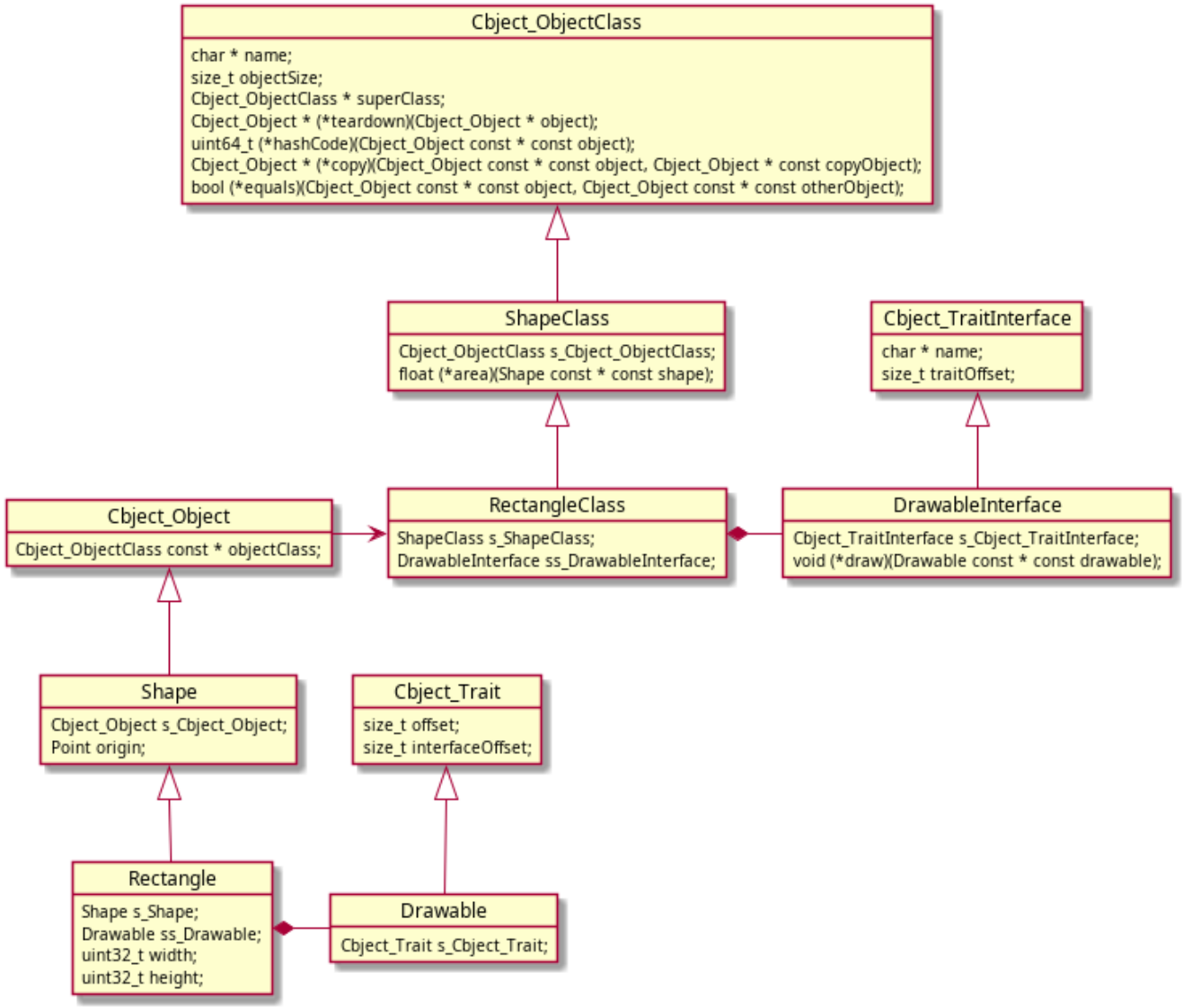

Figure 2. Rectangle class example

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_ObjectClass

```
typedef struct Cbject_ObjectClass Cbject_ObjectClass;
```

Typedef for struct Cbject_ObjectClass

Cbject_Object

```
typedef struct Cbject_Object Cbject_Object;
```

Typedef for struct Cbject_Object

struct Cbject_ObjectClass

```
struct Cbject_ObjectClass {
    char * name;
    size_t objectSize;
    Cbject_ObjectClass const * superClass;
    Cbject_Object * (*teardown)(Cbject_Object * object);
    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);
};
```

Definition of struct Cbject_ObjectClass

Members

- name - Name of the class
- 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 Cbject_Object

```
struct Cbject_Object {
    Cbject_ObjectClass const * class;
};
```

Definition of struct Cbject_Object

Members

- objectClass - Pointer to the class structure

2.1.3. Functions

Cbject_ObjectClass_instance()

```
Cbject_ObjectClass const * Cbject_ObjectClass_instance(void);
```

Get Cbject_ObjectClass instance

Return

Reference of the class instance

Cbject_Object_alloc()

```
Cbject_Object * Cbject_Object_alloc(Cbject_ObjectClass const * const objectClass);
```

Allocate an object in heap memory

Params

- objectClass - Class reference

Return

Reference of the allocated object

Cbject_Object_dealloc()

```
Cbject_Object * Cbject_Object_dealloc(Cbject_Object * const object);
```

Free memory allocated for an object

Params

- object - Cbject_Object reference

Return

NULL

Cbject_Object_init()

```
Cbject_Object * Cbject_Object_init(Cbject_Object * const object);
```

Initialize an object

Params

- object - Cbject_Object reference

Return

Initialized object

Cbject_Object_teardown()

```
Cbject_Object * Cbject_Object_teardown(Cbject_Object * object);
```

Teardown an object.

Params

- object - Cbject_Object reference

Return

NULL

Cbject_Object_copy()

```
Cbject_Object * Cbject_Object_copy(Cbject_Object const * const object,
Cbject_Object * const copyObject);
```

Make a copy of an object.

Params

- object - Cbject_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)

Cbject_Object_equals()

```
bool Cbject_Object_equals(Cbject_Object const * const object, Cbject_Object const
```

```
 * const otherObject);
```

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_Object_hashCode()

```
uint64_t Cbject_Object_hashCode(Cbject_Object const * const object);
```

Get hash code of object

Params

- object - Cbject_Object reference

Return

Cbject_Object hash code

Cbject_Object_isOfClass()

```
bool Cbject_Object_isOfClass(Cbject_Object const * const object,
Cbject_ObjectClass const * const objectClass);
```

Check 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

2.1.4. Tests

test_Cbject_ObjectClass_instance

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 x_Object

Steps

1. Allocate object on stack an initialize it
2. Check if object class points to x_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 x_ObjectClass

Steps

1. Allocate object on stack an initialize it
2. Check if isOfClass method returns true when checked against x_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_Trait

2.2.1. Overview

TODO

2.2.2. Types

Cbject_TraitInterface

```
typedef struct Cbject_TraitInterface Cbject_TraitInterface;
```

Typedef for struct Cbject_TraitInterface

Cbject_Trait

```
typedef struct Cbject_Trait Cbject_Trait;
```

Typedef for struct Cbject_Trait

struct Cbject_TraitInterface

```
struct Cbject_TraitInterface {
    char * name;
    size_t traitOffset;
};
```

Definition of struct Cbject_TraitInterface

Members

- traitOffset - Offset of trait in containing object

struct Cbject_Trait

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

Definition of struct Cbject_Trait

Members

- offset - Offset of Cbject_Trait in container Cbject_Object
- interfaceOffset - Offset of Cbject_TraitInterface in container Cbject_ObjectClass

2.2.3. Functions

Cbject_TraitInterface_instance()

```
Cbject_TraitInterface const * Cbject_TraitInterface_instance(void);
```

Get Cbject_TraitInterface instance

Return

Reference of the trait interface

Cbject_Trait_init()

```
Cbject_Trait * Cbject_Trait_init(Cbject_Trait * const trait);
```

Initialize a trait

Params

- trait - Cbject_Trait reference

Return

Initialized trait

2.3. Cbject_Utils

2.3.1. Overview

TODO

2.3.2. Types

Cbject_Any

```
typedef void Cbject_Any;
```

Typedef for Cbject_Any

Remark

To be used with pointers to anything

2.3.3. Macros

Cbject_typedefClass()

```
#define Cbject_typedefClass(className)
```

Syntactic sugar to define types for a class

Params

- className - Name of the class

Cbject_setUpClass()

```
#define Cbject_setUpClass(className, superClassName, objectClass)
```

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

Params

- className - Name of the class
- superClassName - Name of the super class
- objectClass - Class instance

Cbject_bindClassMethod()

```
#define Cbject_bindClassMethod(className, methodName, objectClass)
```

Bind a method of a class

Params

- className - Name of the class
- methodName - Name of the method
- objectClass - Class instance

Cbject_setUpInterfaceOf()

```
#define Cbject_setUpInterfaceOf(className, interfaceName, objectClass)
```

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

Params

- className - Name of the class
- interfaceName - Name of the interface
- objectClass - Class instance

Cbject_bindInterfaceMethodOf()

```
#define Cbject_bindInterfaceMethodOf(className, interfaceName, methodName,
objectClass)
```

Bind a method of an interface

Params

- className - Name of the class
- interfaceName - Name of the interface
- methodName - Name of the method
- objectClass - Class instance

Cbject_alloc()

```
#define Cbject_alloc(className)
```

Syntactic sugar to allocate an object in heap memory

Params

- className - Name of class

Return

Reference of the allocated object

Cbject_dealloc()

```
#define Cbject_dealloc(object)
```

Syntactic sugar to free memory allocated for an object

Params

- object - Cbject_Object reference

Return

NULL

Cbject_init()

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

Syntactic sugar for object initialization

Params

- className - Name of the class

- ...
  - object - Cbject_Object reference
  - ... - Init params

Return

Initialized object

Cbject_setUpObject()

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

Cbject_Object setup (initialize, set the object class)

Params

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

Cbject_setUpTraitOf()

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

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

Params

- className - Name of the class
- interfaceName - Name of the interface
- ...
  - object - Cbject_Object reference
  - ... - Init params

Cbject_allocInit()

```
#define Cbject_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

Cbject_sallocInit()

```
#define Cbject_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

Cbject_teardown()

```
#define Cbject_teardown(object)
```

Syntactic sugar to teardown an object.

Params

- object - Cbject_Object reference

Return

NULL

Cbject_copy()

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

Syntactic sugar to make a copy of an object.

Params

- className - Name of class
- object - Cbject_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)

Cbject_allocCopy()

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

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

Params

- className - Name of class
- object - Cbject_Object reference

Return

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

Cbject_sallocCopy()

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

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

Params

- className - Name of class
- object - Cbject_Object reference

Return

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

Cbject_equals()

```
#define 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_hashCode()

```
#define Cbject_hashCode(object)
```

Syntactic sugar to get hash code of object

Params

- object - Cbject_Object reference

Return

Cbject_Object hash code

Cbject_isOfClass()

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

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

Params

- object - Cbject_Object reference
- className - Class name

Return

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

Cbject_typedefInterface()

```
#define Cbject_typedefInterface(interfaceName)
```

Syntactic sugar to define types for an interface

Params

- interfaceName - Name of the interface

Cbject_setUpInterface()

```
#define Cbject_setUpInterface(interfaceName, traitInterface)
```

Interface setup (initialize super)

Params

- interfaceName - Name of the interface
- traitInterface - Interface instance

Cbject_bindInterfaceMethod()

```
#define Cbject_bindInterfaceMethod(interfaceName, methodName, traitInterface)
```

Bind a method of an interface

Params

- interfaceName - Name of the interface
- superInterfaceName - Name of the super interface
- methodName - Name of the method
- traitInterface - Interface instance

Cbject_initTrait()

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

Syntactic sugar for trait initialization

Params

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

Return

Initialized trait

Cbject_doOnce

```
#define Cbject_doOnce
```

Run a block of code only once

Usage

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

Remark

Not thread safe

Cbject_assertStatic()

```
#define Cbject_assertStatic(expression, identifier)
```

Compile time assert

Params

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

Cbject_castTo()

```
#define Cbject_castTo(typeName, instance)
```

Cast an instance to the provided typeName

Params

- typeName - Name of the type (class or interface)
- instance - Instance to cast

Return

Instance cast to the provided typeName

Cbject_lengthOf()

```
#define Cbject_lengthOf(array)
```

Get length of an array

Params

- array - Array for which to get the length

Cbject_salloc()

```
#define Cbject_salloc(typeName)
```

Syntactic sugar to allocate memory on the stack

Params

- typeName - Name of type

Return

Reference of the allocated memory

Cbject_ignore()

```
#define Cbject_ignore(var)
```

Syntactic sugar to ignore unused variables

Params

- var - Variable to be ignored

Cbject_is()

```
#define Cbject_is(typeName)
```

Syntactic sugar to extend a type

Remark

Should be used as the first member in the structure

Params

- typeName - Name of the type

Cbject_has()

```
#define Cbject_has(typeName)
```

Syntactic sugar to compose a type with the provided typeName

Remark

Should be used after Cbject_is() macro

Params

- typeName - Name of the type

Cbject_class()

```
#define Cbject_class(className)
```

Syntactic sugar to get class reference

Params

- className - Name of the class

Return

Class reference

Cbject_singleton()

```
#define Cbject_singleton(className)
```

Syntactic sugar to get a singleton reference

Params

- className - Name of the class

Return

Singleton reference

Cbject_classOf()

```
#define Cbject_classOf(object)
```

Get the class of an object

Params

- object - Cbject_Object reference

Return

Class reference

Cbject_objectSizeOf()

```
#define Cbject_objectSizeOf(object)
```

Get the size in memory of an object

Params

- object - Cbject_Object reference

Return

Cbject_Object size

Cbject_traitOf()

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

Get trait of an object

Params

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

Return

Cbject_Trait reference

Cbject_callObjectMethod()

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

Call a method through an object

Params

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

Return

Depends on the called method

Cbject_callClassMethod()

```
#define Cbject_callClassMethod(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 - Cbject_Object reference
  - ... - Method params

Return

Depends on the called method

Cbject_offsetOf()

```
#define Cbject_offsetOf(trait)
```

Get offset of a trait in container object

Params

- trait - Cbject_Trait reference

Return

Offset of trait in container object

Cbject_interfaceOffsetOf()

```
#define Cbject_interfaceOffsetOf(trait)
```

Get the interface offset in container class

Params

- trait - Cbject_Trait reference

Return

Offset of interface in container class

Cbject_objectOf()

```
#define Cbject_objectOf(trait)
```

Get container object from a trait

Params

- trait - Cbject_Trait reference

Return

Reference of the container object

Cbject_interfaceOf()

```
#define Cbject_interfaceOf(trait)
```

Get the interface of a trait

Params

- trait - Cbject_Trait reference

Return

Interface reference

Cbject_interface()

```
#define Cbject_interface(interfaceName)
```

Syntactic sugar to get interface reference

Params

- interfaceName - Name of the interface

Return

Interface reference

Cbject_callTraitMethod()

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

Call a method through a trait

Params

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

Return

Depends on the called method

Cbject_callInterfaceMethod()

```
#define Cbject_callInterfaceMethod(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 - Cbject_Trait reference
  - ... - Method params

Return

Depends on the called method

Cbject_VaArgs_first()

```
#define Cbject_VaArgs_first(...)
```

Get first argument from *VA_ARGS*

Params

- ... - *VA_ARGS*

Cbject_VaArgs_rest()

```
#define Cbject_VaArgs_rest(...)
```

Get list of arguments from *VA_ARGS* except the first

Remark

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

Params

- ... - *VA_ARGS*

2.4. Cbject_Settings

2.4.1. Overview

TODO

2.4.2. Macros

Cbject_useShortNames

```
#define Cbject_useShortNames ...
```

Setting to configure the use of short names (eg: Cbject_Object → x_Object)

Values

- true - Use short names
- false - Use long names