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_Object_dealloc(object);
```

1.3. cbject_Object model

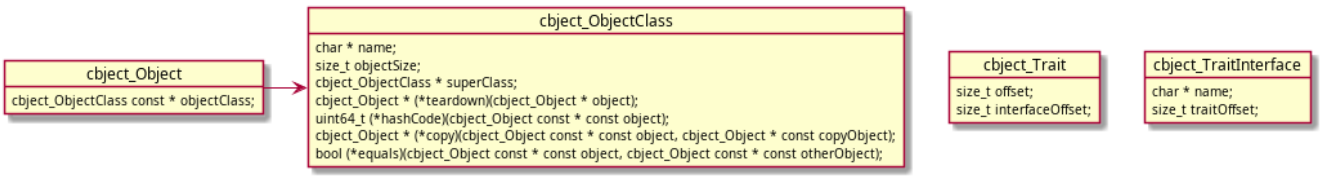

Figure 1. Building blocks

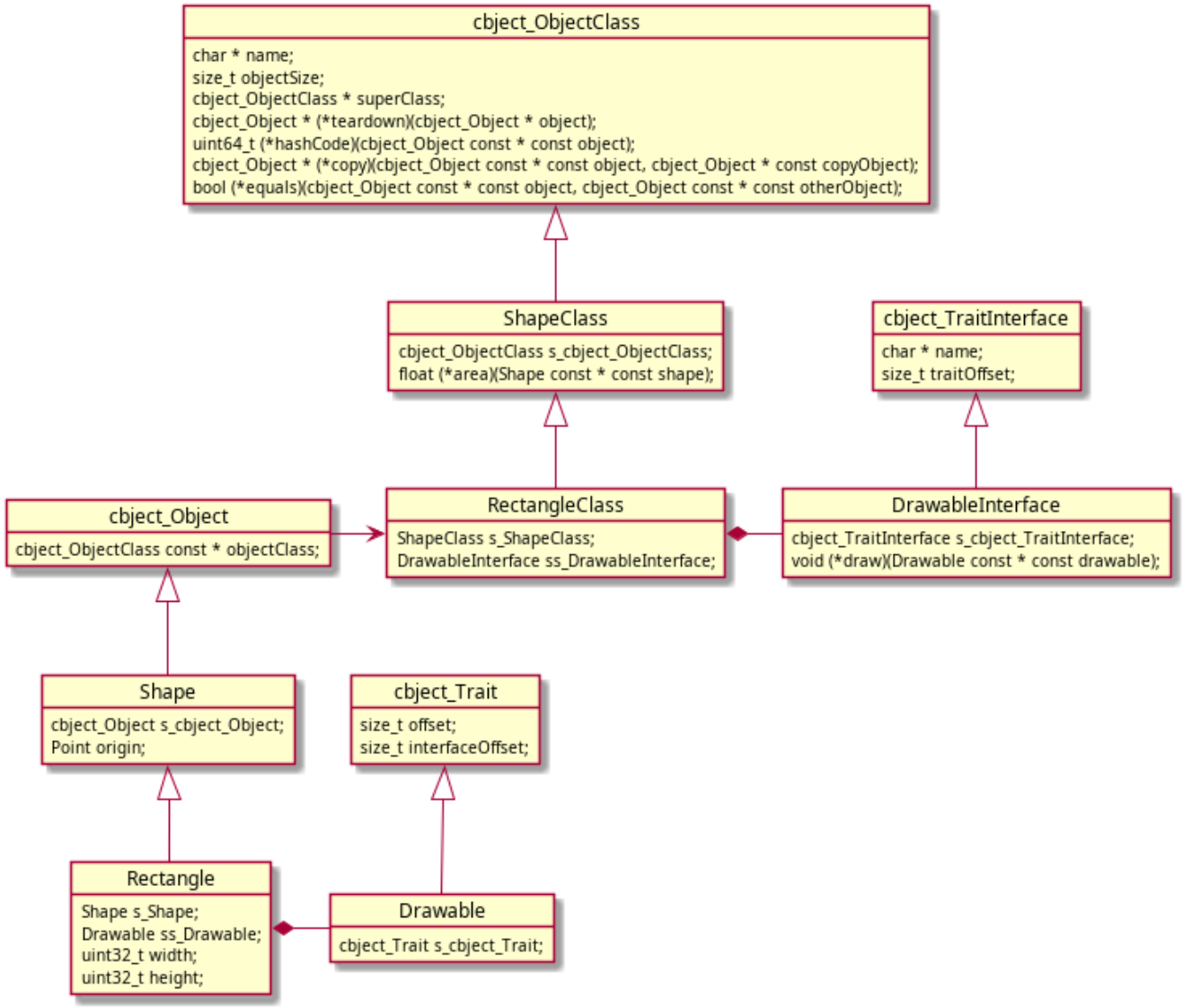

Figure 2. Rectangle class example

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_Object, cbject_ObjectClass

```
cbject_utils_typedefClass(cbject_Object);
```

Typedef for struct cbject_ObjectClass and 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_getInstance()

```
cbject_ObjectClass const * cbject_ObjectClass_getInstance(void);
```

Get cbject_ObjectClass instance

Return

Reference of the class instance

cbject_ObjectClass_alloc()

```
cbject_Object * cbject_ObjectClass_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_getInstance

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_Trait

2.2.1. Overview

TODO

2.2.2. Types

cbject_Trait, cbject_TraitInterface

```
cbject_utils_typedefInterface(cbject_Trait);
```

Typedef for struct cbject_Trait and struct cbject_TraitInterface

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_getInstance()

```
cbject_TraitInterface const * cbject_TraitInterface_getInstance(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. Macros

cbject_utils_typedefClass()

```
cbject_utils_typedefClass(className)
```

Syntactic sugar to define types for a class

Params

- className - Name of the class

cbject_utils_typedefInterface()

```
cbject_utils_typedefInterface(interfaceName)
```

Syntactic sugar to define types for an interface

Params

- interfaceName - Name of the interface

cbject_utils_cast()

```
cbject_utils_cast(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_getClassOfObject()

```
cbject_getClassOfObject(object)
```

Get the class of an object

Params

- object - cbject_Object reference

Return

Class reference

cbject_getSizeOfObject()

```
cbject_getSizeOfObject(object)
```

Get the size in memory of an object

Params

- object - cbject_Object reference

Return

cbject_Object size

cbject_getTraitOfObject()

```
cbject_getTraitOfObject(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_callMethodOfObject()

```
cbject_callMethodOfObject(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_callMethodOfClass()

```
cbject_callMethodOfClass(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_utils_getInterfaceOffsetOfTrait()

```
cbject_utils_getInterfaceOffsetOfTrait(trait)
```

Get the interface offset in container class

Params

- trait - cbject_Trait reference

Return

Offset of interface in container class

cbject_getObjectOfTrait()

```
cbject_getObjectOfTrait(trait)
```

Get container object from a trait

Params

- trait - cbject_Trait reference

Return

Reference of the container object

cbject_utils_getInterfaceOfTrait()

```
cbject_utils_getInterfaceOfTrait(trait)
```

Get the interface of a trait

Params

- trait - cbject_Trait reference

Return

Interface reference

cbject_callMethodOfTrait()

```
cbject_callMethodOfTrait(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_callMethodOfInterface()

```
cbject_callMethodOfInterface(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_utils_VaArgs_first()

```
cbject_utils_VaArgs_first(...)
```

Get first argument from *VA_ARGS*

Params

- ... - *VA_ARGS*

cbject_utils_VaArgs_rest()

```
cbject_utils_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_settings_useShortNames

```
cbject_settings_useShortNames ...
```

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

Values

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