# Objclint Documentation

*Release 0.1*

Alexander Smirnov

October 18, 2016

# Contents

Objclint is a clang based tool for performing code style guidelines check for Objective-C project. Each rule is just JS script.

**Author** Alexander Smirnov. alexander [at] smirn0v.ru

**Version** 0.1 Alpha. Try on your own risk.

**Github** https://github.com/smirn0v/objclint

Contents:

CHAPTER 1

# Installation

Run following commands in terminal to install objclint

```
$ git clone git@github.com:smirn0v/objclint.git
$ cd objclint
$ ./install.sh
```

CHAPTER 2

# Usage

## 2.1 Simple usage

Objclint currently is only usable via terminal. One command you will need is

```
$ objclint-xcodebuild
```

It just proxy to **xcodebuild** command and accept any argument that **xcodebuild** accept.

## 2.2 Configuration file

By default **objclint** will use *./lints* folder for JS validation scripts and will analyze all files passed into during **xcodebuild** process. But it is possible to alter this behaviour, create **.objclint** file in the same folder as project file is. Here is example of possible configuration file

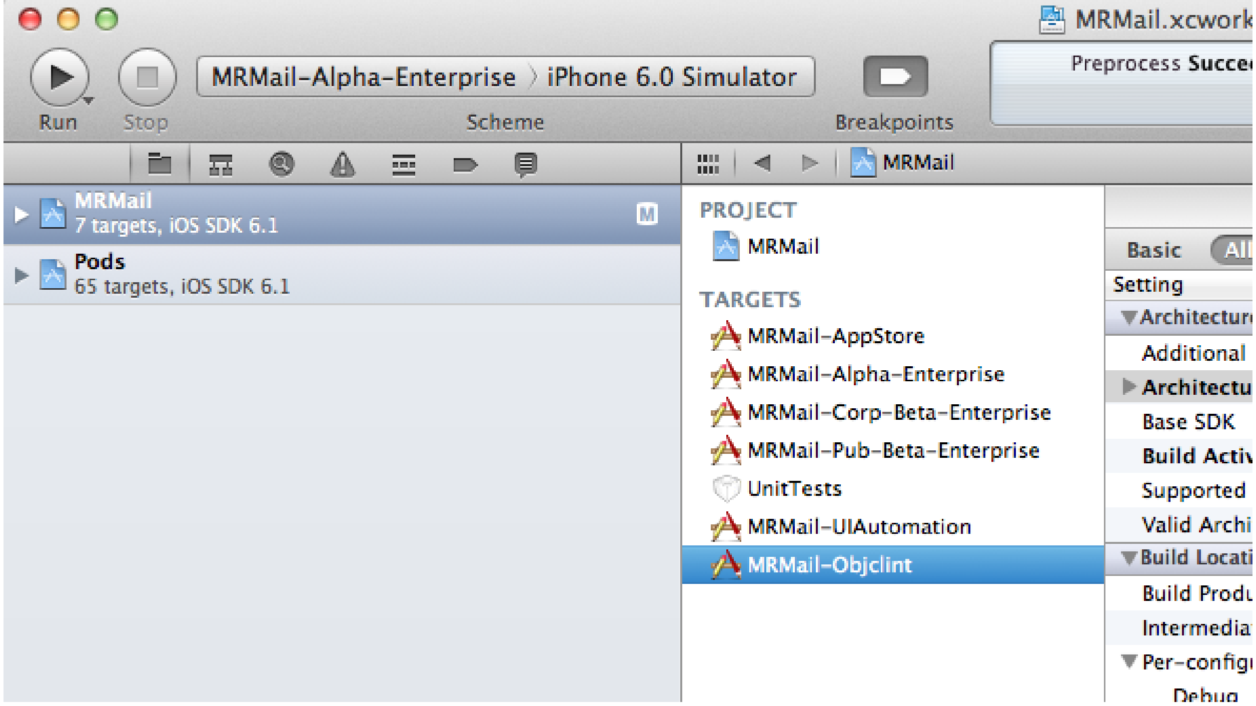
```
{
    "lints-directories": ["./my-lints"],
    "ignores": [
        "src/infrastructure/blowfish/ablowfishv1.h$",
        "local-pods/.*",
        "Pods/.*",
        "thirdparty/.*"
    ]
}
```

So **objclint** will look for JS lints in *./my-lints* folder and will ignore some files in listed folders.

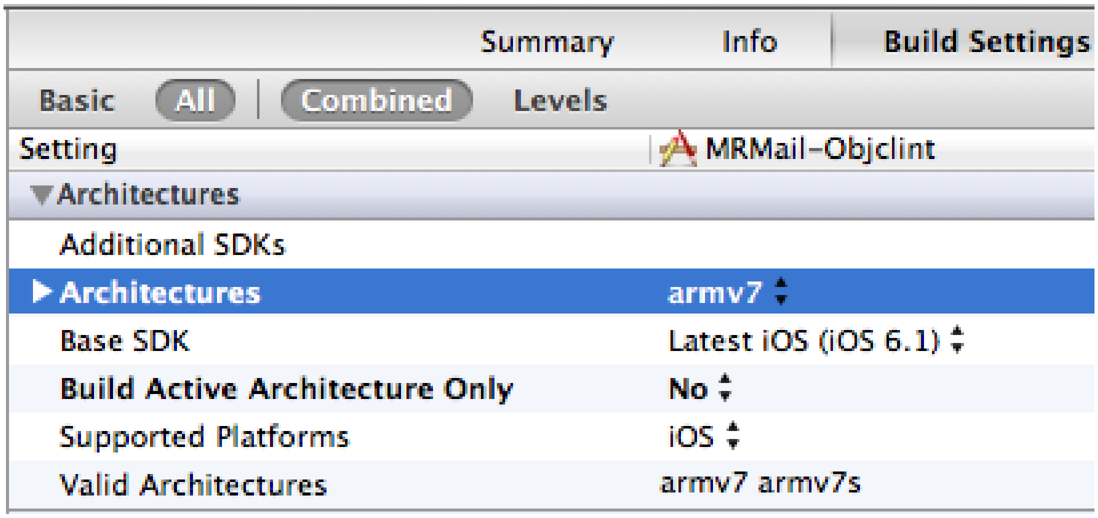
## 2.3 Project configuration

You should create seperate project target to ignore any resource file and don't perform analyzis twice for every supported platform.

### 2.3.1 Seperate target ‘Objclint’ copied from your main target.

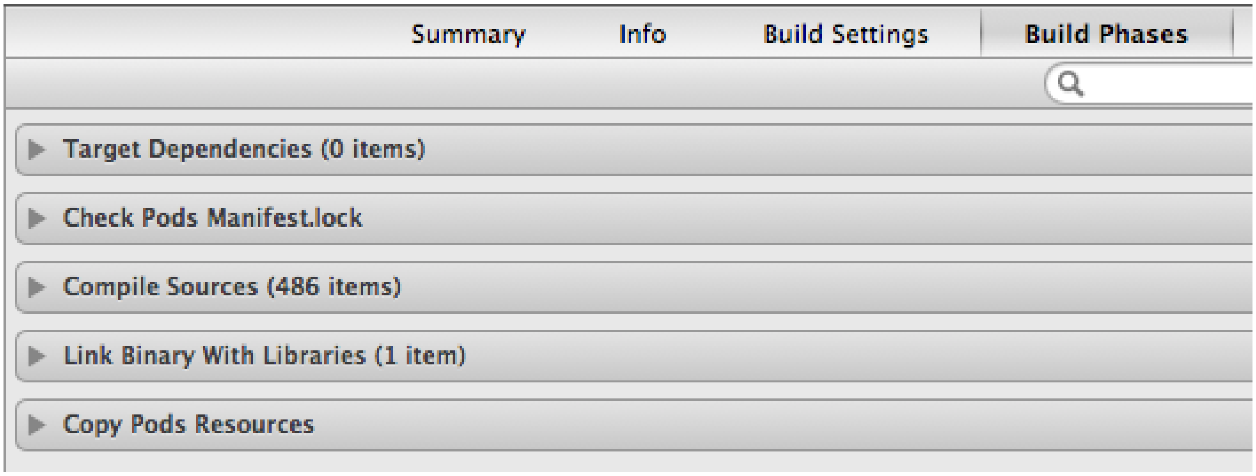


### 2.3.2 Only one architecture to build for.



| Setting | MRMail-Objclint |
|---|---|
| ▼Architectures | |
| Additional SDKs | |
| ▶ Architectures | armv7 |
| Base SDK | Latest iOS (iOS 6.1) |
| Build Active Architecture Only | No |
| Supported Platforms | iOS |
| Valid Architectures | armv7 armv7s |

### 2.3.3 No unnecessary build phases.



Target Dependencies (0 items)

Check Pods Manifest.lock

Compile Sources (486 items)

Link Binary With Libraries (1 item)

Copy Pods Resources

CHAPTER 3

# JS API

All **objclint** analyzis performed with JavaScript.

```
var objc_msg = cursor.kind=="ObjCMessageExpr";
var nsassert_method = cursor.spelling == "handleFailureInMethod:object:file:lineNumber:description:";

if(objc_msg && nsassert_method) {
    lint.reportError("Using NSAssert() may cause retin cycles. Try NSCAssert() instead");
}
```

There are two global objects:

1. `cursor`
2. `lint`

And number of other objects that you can get:

1. `token`
2. `objc-method-declaration`

## 3.1 Cursor object

Cursor is one of the most significant objects in objclint. Clang builds Abstract Syntax Tree of every source file and allows us to visit every element of it as **cursor** and its childs. Every time javascript validator called you have global **cursor** object that can be analyzed.

**cursor** have following properties:

**`lineNumber`** Line number of **cursor**.

**`column`** Line column of **cursor**.

**`offset`** Global offset in file.

**`fileName`** Current file name.

**`displayName`** Retrieve the display name for the entity referenced by this cursor.

The display name contains extra information that helps identify the cursor, such as the parameters of a function or template or the arguments of a class template specialization.

**`USR`** Retrieve a Unified Symbol Resolution (USR) for the entity referenced by the given cursor.

A Unified Symbol Resolution (USR) is a string that identifies a particular entity (function, class, variable, etc.) within a program. USRs can be compared across translation units to determine, e.g., when references in one translation refer to an entity defined in another translation unit.

**`spelling`** Retrieve a range for a piece that forms the cursors spelling name. Most of the times there is only one range for the complete spelling but for objc methods and objc message expressions, there are multiple pieces for each selector identifier.

**`kind`** Retrieve the kind of the given cursor. See clang documentation for full list (ignore `CXCursor` prefix).