

iOS Project bootstrap

How do you setup your iOS projects?

Since we are approaching 2015 I'm working on refreshing my project bootstrap. I've decided to open source it so other can benefit or contribute.

I think it's pretty neat but decide for yourself.

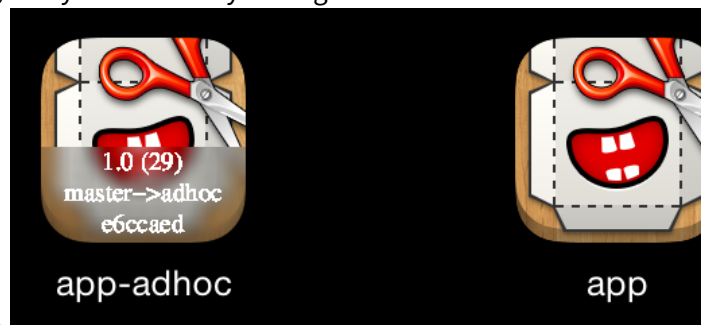
It's still WIP so pull requests are most welcomed.

Basic setup and configuration

A project by default has 3 configurations:

1. Debug
2. Release
3. Adhoc

Each configuration can be put side by side on same device, and you can clearly distinguish each build.



Easier to find issues across different versions and branches.

Looking at each of the icon you get following informations: - Build number: 29 - Branch: Master - Commit hash - Version: 1.0 - Configuration that the app was built with

You can also use KZBootstrap API to query those informations while running your application.

```
2014-10-15 14:29:41:716 [INF] KZAppDelegate.m:29 (-[KZAppDelegate application:didFinishLaunchingWithOptions:]): KZBootstrap:
shortVersion: 1.0
branch: master
buildNumber: 3
environment: STAGING
```

Code Quality and Warnings

Warnings were added by compiler team for a reason, as such I start with Weverything and disable few warnings: - Wno-objc-missing-property-synthesis - don't want to do @synthesize on properties - Wno-unused-macros - annoying when doing DSL - Wno-disabled-macro-expansion - needed for DSL/Metaprogramming - Wno-gnu-statement-expression - helpful - Wno-language-extension-token - language extensions are useful - Wno-overriding-method-mismatch - so I can change id to specific type and avoid unnecessary local variables

Also treat warnings as errors is a must.

That's not all, let's add some scripts:

- turn all todo/fixme into warnings `//! TODO: example`
- warnings when files get too big `File /Users/merowing/Dropbox/Programowanie/Projekty/Bootstrap/Example/KZBootstrap/KZViewController.m has more than 250 lines (267), consider refactoring`
 - add `KZBIgnoreLineCount` anywhere in file to disable warning generation for that file.
- Automatically generate macro for current developer, that way a team can have different code paths while they are working on features, or different logging levels. Without git changes.

```
1  #if merowing
2  //! my code
3  #endif
```

One more thing, let's add some macros: To prevent nil passed in as arguments:

- `KZB_REQUIRE_ALL_PARAMS`
- `KZB_REQUIRE_PARAMS(1,2)`

```
[self doSomethingWithCompletion:nil];
```

Null passed to a callee which requires a non-null argument 2

When your subclasses should call super: `* KZB_REQUIRE_SUPER`

```
The 'viewDidLoad' instance method in UIViewController subclass 'KZViewController' is missing a [super viewDidLoad] call
```

When you want to avoid spelling errors: `* KZB_KEYPATH * KZB_KEYPATH_T`

```
DDLogInfo(@"keyPath %@", KZB_KEYPATH(window.windowLevel));
```

Property 'windowLevels' not found on object of type 'UIWindow *'; did you mean 'windowLevel'?

Issue

Fix-it Replace "windowLevels" with "windowLevel"

Environments

Often when working with big clients, you need to have multiple environments for Staging / Production / QA etc. They usually differ in some kind of configuration, eg. different URL endpoints.

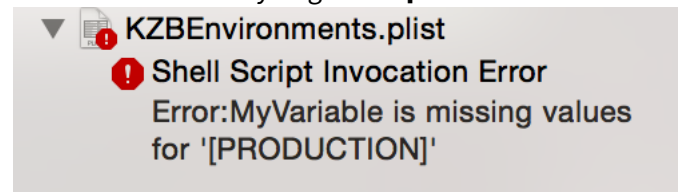
Too many times I've seen people creating separate targets for each of them, which leads to maintenance costs and unnecessary bloat/clutter.

As such I've created a different approach, with some nice automation: - Default environments can be changed either via xcodebuild user variable (on Jenkins) or via launch argument inside your schema. - on jenkins, xcodebuild ... KZBEnv=@\"Production\" build - in custom Schema: Edit Scheme->Arguments->Launch Arguments-> \"-KZBEnvOverride Production\" - Environments can be changed without reinstalling application, even while it's running. - All environments variables are created in a

Key	Type	Value
▼ Root	Dictionary	(2 items)
▼ KZBEnvironments	Array	(3 items)
Item 0	String	QA
Item 1	String	STAGING
Item 2	String	PRODUCTION
▼ MyVariable	Dictionary	(3 items)
QA	String	Something
STAGING	String	SomethingElse
PRODUCTION	String	Prod!!

single plist

- If any of the variables are missing an entry for one of the environments you get **compile time error**.



You can even click on it to go to the configuration file.

< app-dev

QA

STAGING

PRODUCTION

- Settings bundle will be *automatically injected* to give you environment switching.

- You can register for callback when env changes, useful if you need to reset your database etc. - Production builds will remove all variables for other environments to prevent exposing non-production and unused configurations.

Logging - Optional

If you are using CocoaLumberjack you can include KZBootstrap/Logging subspec to get log formatting that works as clickable links in AppCode.

```
2014-10-14 15:40:01:799 [INF] User.m:97 (-[User signalForDailyEntries]): Querying data up
2014-10-14 15:40:14:779 [INF] User.m:103 (__29-[User signalForDailyEntries]_block_invoke)
```

Debugging - Optional

If you decide to include KZBootstrap/Debug subspec, you will get:

- assertions when UIKit is layouted/displayed on background thread, so you can fix your code.
- API interception capabilities for AFNetworking, which you can either display yourself (or send me PR with universal UI). Or just look at during debugging by calling

```
1 [KZBResponseTracker printAll]
```

```
(lldb) po [KZBResponseTracker printAll]
2014-10-15 19:18:03.157 KZBootstrap[12041:1725007] endPoint http://api.openweathermap.org/data/2.5/weather?lat=35&lon=139 has been called 1 times:
2014-10-15 19:18:03.158 KZBootstrap[12041:1725007] [0] 10/15/14, 7:18:00 PM {
response <NSURLSessionResponse: 0x7b78cdeb> { URL: http://api.openweathermap.org/data/2.5/weather?lat=35&lon=139 } { status code: 200, headers {
    "Access-Control-Allow-Credentials" = true;
    "Access-Control-Allow-Methods" = "GET, POST";
    "Access-Control-Allow-Origin" = "*";
    Connection = "keep-alive";
    "Content-Type" = "application/json; charset=utf-8";
    Date = "Wed, 15 Oct 2014 17:17:59 GMT";
    Server = nginx;
    "Transfer-Encoding" = Identity;
    "X-Source" = redis;
} }
object {
    base = "cmc stations";
    clouds = {
        all = 92;
    };
    cod = 200;
```

Installing KZBootstrap

KZBootstrap is available through CocoaPods. To install it, simply add the following line to your Podfile:

- KZBootstrap - for core functionality
- KZBootstrap/Logging - additional logging functionality
- KZBootstrap/Debug - additional debugging functionality

There are few things you need to do with your project, you can either use my crafter setup tool to make it automatic or do it manually:

- KZBEnvironments.plist with KZBEnvironments key containing array of all environments. All extra keys are treated as env variables and should have a value for each of the allowed env's.
- BUNDLE_DISPLAY_NAME_SUFFIX _and BUNDLE_ID_SUFFIX should be added to User-Defined settings
 - In your target .plist file append both display name and bundle identifier keys with those variables eg. `app${BUNDLE_DISPLAY_NAME_SUFFIX}`
- Add KZBEnv user-defined setting with value of default env for each configuration then in pre-processor macros add `KZBDefaultEnv=${KZBEnv}`
- Add empty file named KZBootstrapUserMacros.h anywhere in your project, and include it into your *prefix.pch file. Include that file in your .gitignore.
- Set warnings as described above.
- You should have Settings.bundle in the project so the code can inject it with environment switching functionality.
- Add script execution at the end of your Build Phases `"${SRCROOT}/Pods/KZBootstrap/Pod/Assets/Scripts/bootstrap.sh"` You can selectively enable features by passing arguments to bootstrap.sh
 - `-l` will enable line-count warnings
 - `-t` will enable TODO warnings
 - `-u` will enable user macros
 - `-n` will enable build-number automation
 - `-i` will enable icon versioning. (This also enables build-number automation.)

Base crafter setup might look like this, replace CUSTOM with your preferred steps:

```

1 # All your configuration should happen inside configure block
2 Crafter.configure do
3
4   # This are projects wide instructions
5   add_platform({:platform => :ios, :deployment => 7.0})
6   add_git_ignore
7   duplicate_configurations({:adhoc => :release})
8
9   # set of options, warnings, static analyser and anything else normal
   #   xcde treats as build options
10  set_options %w(
11    RUN_CLANG_STATIC_ANALYZER
12    GCC_TREAT_WARNINGS_AS_ERRORS
13  )
14
15  set_build_settings({
16    : 'WARNING_CFLAGS' => %w(
17      -Weverything

```

```

18     -Wno-objc-missing-property-synthesis
19     -Wno-unused-macros
20     -Wno-disabled-macro-expansion
21     -Wno-gnu-statement-expression
22     -Wno-language-extension-token
23     -Wno-overriding-method-mismatch
24     ).join(" ")
25 })
26
27 set_build_settings({
28     :BUNDLE_ID_SUFFIX => '.dev',
29     :BUNDLE_DISPLAY_NAME_SUFFIX => 'dev',
30     :KZBEnv => 'QA'
31 }, configuration: :debug)
32
33 set_build_settings({
34     :BUNDLE_ID_SUFFIX => '.adhoc',
35     :BUNDLE_DISPLAY_NAME_SUFFIX => 'adhoc',
36     :KZBEnv => 'QA'
37 }, configuration: :adhoc)
38
39 set_build_settings({
40     :BUNDLE_ID_SUFFIX => '',
41     :BUNDLE_DISPLAY_NAME_SUFFIX => '',
42     :KZBEnv => 'PRODUCTION'
43 }, configuration: :release)
44
45 # CUSTOM: Modify plist file to include suffix and displayname
46 # CUSTOM: Add empty KZBootstrapUserMacros.h file to your project and
47 # CUSTOM: Add KZBEnvironments.plist with list of your environments
48 # CUSTOM: Add KZBEnvironments.plist with list of your environments
49 # CUSTOM: Add KZBEnvironments.plist with list of your environments
50 # CUSTOM: Add KZBEnvironments.plist with list of your environments
51 # CUSTOM: Add KZBEnvironments.plist with list of your environments
52 # CUSTOM: Add KZBEnvironments.plist with list of your environments
53 # CUSTOM: Add KZBEnvironments.plist with list of your environments
54 # CUSTOM: Add KZBEnvironments.plist with list of your environments
55 # CUSTOM: Add KZBEnvironments.plist with list of your environments
56 # CUSTOM: Add KZBEnvironments.plist with list of your environments
57 # CUSTOM: Add KZBEnvironments.plist with list of your environments
58 # CUSTOM: Add KZBEnvironments.plist with list of your environments
59 # CUSTOM: Add KZBEnvironments.plist with list of your environments

```

In you want to support dynamic env switching app delegate you can add something like this:

```

1  NSLog(@"user variable = %@, launch argument %@", @"d", [[

```

```
    [NSUserDefaults standardUserDefaults] objectForKey:@"KZBEnvOverride"
    ]);
2   KZBootstrap.defaultBuildEnvironment = KZBEnv;
3   KZBootstrap.onCurrentEnvironmentChanged = ^(NSString *newEnv,
    NSString *oldEnv) {
4       NSLog(@"Changing env from %@ to %@", oldEnv, newEnv);
5   };
6   [KZBootstrap ready];
7
8   NSLog(@"KZBootstrap:\n\tshortVersion: %@\n\tbranch: %@\n\tbuildNumber
    : %@\n\tenvironment: %@", KZBootstrap.shortVersionString,
    KZBootstrap.gitBranch, @(KZBootstrap.buildNumber), KZBootstrap.
    currentEnvironment);
```

License

KZBootstrap is available under the MIT license. See the LICENSE file for more info.

Author

Krzysztof Zablocki, krzysztof.zablocki@pixle.pl

My blog

Follow me on twitter.

Attributions

All of this wouldn't be possible if we didn't have such a great community, based on my own previous work but also countless others. Tried to reference everything but if you think I missed something please let me know.

References:

<http://stackoverflow.com/questions/10497552/how-to-configure-independent-sets-of-runtime-settings-in-xcode>

<https://github.com/crushlovely/Amaro>

<https://github.com/crushlovely/Sidecar>

<http://swwritings.com/post/2013-05-20-concurrent-debug-beta-app-store-builds>

<https://gist.github.com/dulaccc/a52154ac4c007db2be55>

<https://gist.github.com/steipete/5664345>

<http://blog.manbolo.com/2013/05/17/passing-user-variable-to-xcodebuild>

<http://blog.jaredsinclair.com/post/97193356620/the-best-of-all-possible-xcode-automated-build>

<https://github.com/krzysztofzablocki/crafter>

<https://github.com/krzysztofzablocki/IconOverlaying>

Big thanks goes to Lextech Global Services for supporting my community work, they are awesome. If you need a mobile app for your enterprise, you should talk to them.