Debugging

IRON HACK

Leo Font

LLDB Using Breakpoints

- Adding a breakpoint
- The debug navigator
- Debug area with variables view
- Stepping through code
- Deleting breakpoints

Exception Breakpoint

- From Xcode’s BreakPoint Navigator
- You tell the debugger to break automatically on any line that causes your application to crash.

Symbolic Breakpoint

- From Xcode’s BreakPoint Navigator
- Stops execution when certain message is send
  -[NSObject doesNotRecognizeSelector:]

Editing breakpoints

- Can set conditions, actions and options.
- **frame variable var** shows local variable within the current stack frame when the breakpoint hits
- **frame variable self->var** shows local variable within the current stack frame when the breakpoint hits

LLDB Console

- **po** = Print Object == [object description]
- Use **p** for primitive types and C structs.
- **breakpoint list** = List breakpoints
- **breakpoint delete** = Delete breakpoints
- **breakpoint set -r expr -s Module**

Allows to create breakpoints using expressions!
^For our example breakpoint set calc -s ProjectName

LLDB Console-II

- **expression** allows to view and change values
- expression self->memValue
- expression self->memValue = 25.0
- You can edit/change values in memory (in variable editor)
- If you need help use **help** & **apropos**

Debugging views

- Views can be debugged in a 3D representation using the "Debug View Hierarchy option"

Static Analyzer

- Analizes code without need to run it
- Examines each function and method iterating over every possible path.
- Solves “logic” problems.

```
-(int)tellMyAge:(BOOL)shouldTellAge{
      int age;
      if (shouldTellAge) {
        age = 40;
      }
      return age;
    }
```

Compiler Config Flags

- Compiler warns code errors but also dangerous patterns
- Xcode has a default set of conservative warnings enabled
- Warnings can be enabled/disabled using Xcode’s Project Build Setting UI
- You can also enable **all** warnings setting compiler flags (not in UI)

Compiler Config Flags

- You have to add one or more custom build flags to the Other Warning Flags build setting
- The compiler flag -W… lets you to enable or disable specific warnings as well as predefined sets
- -Wall -Wno-unused-variable tell the compiler, “enable ‘all’ warnings but don’t warn me about unused variables”.

Compiler Config Flags

-Wall

This sounds like a set of all warnings, but it actually only covers a subset of the available warnings. Will provide you with quite an aggressive warning level that will probably point out a bunch of potential errors in your code that the default settings did not catch

Compiler Config Flags

-Wextra

This grouping includes warnings that can be equally useful as those in -Wall, but may trigger a higher false-positive rate or common philosophical objections. A good example of this is the -Wsign-compare flag, which warns you when you compare a signed integer to an unsigned type.

Compiler Config Flags

-Weverything

-This is the magic incantation that actually enables every warning in Clang, including those that may still be in development and contain bugs. Don’t use this on your code.

Compiler Config Flags

- There's no magic recipe. Start with -Wall & -Wextra and **CHECK YOUR BUILD LOG**
- You can also disable a warning temporarily

~~~
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wsign-compare"

```objectivec
int i = 100;
unsigned int u = 200;
if (i < u) {  // no warning
  …
}

#pragma clang diagnostic pop
~~~

Check

http://fuckingclangwarnings.com

Targets

- A project always has at least one target. When you build and run, you build and run the target, not the project
- Product that target builds may be an App, library or test.
- Targets can be seen at project’s navigation list
- You can duplicate your target or add newer ones

Targets

- Duplicate a target + modify scheme + Build settings + preprocessor settings + plist
- Specify assets’ target if required
- Identifying at compile time:

```
#if defined(LITE)
  NSLog(@"Lite version");
#else
  NSLog(@"Original version");
#endif
```

Document generation with appledoc

- See https://github.com/tomaz/appledoc
- Install appledoc:
  - CD your projects directory
  - Download: git clone git://github.com/tomaz/appledoc.git
  - Install: sudo sh install-appledoc.sh
- Integrate it with Xcode:

Sample script for document generation with appledoc

```
/usr/local/bin/appledoc \
--project-name "${PROJECT_NAME}" \
--project-company “IronHack” \
--company-id "com.ironHack” \
--output “~/dev2015/Help/${PROJECT_NAME}" \
--install-docset \
--logformat xcode \
--keep-undocumented-objects \
--keep-undocumented-members \
--keep-intermediate-files \
--no-repeat-first-par \
--no-warn-invalid-crossref \
--merge-categories \
--exit-threshold 2 \
--docset-platform-family iphoneos \
--ignore "*.m" \
--index-desc "${PROJECT_DIR}/readme.md” \
"${PROJECT_DIR}/${PROJECT_NAME}" \
```

Document generation with appledoc

- Create readme.md file & set target Documentation, not project target.
- Select Documentation target and build
- Comment format:
  /** This view controller is the main controller of this application */
  @interface ViewController : UIViewController
- Markdown available (bullet list *, links, sections #, tables)

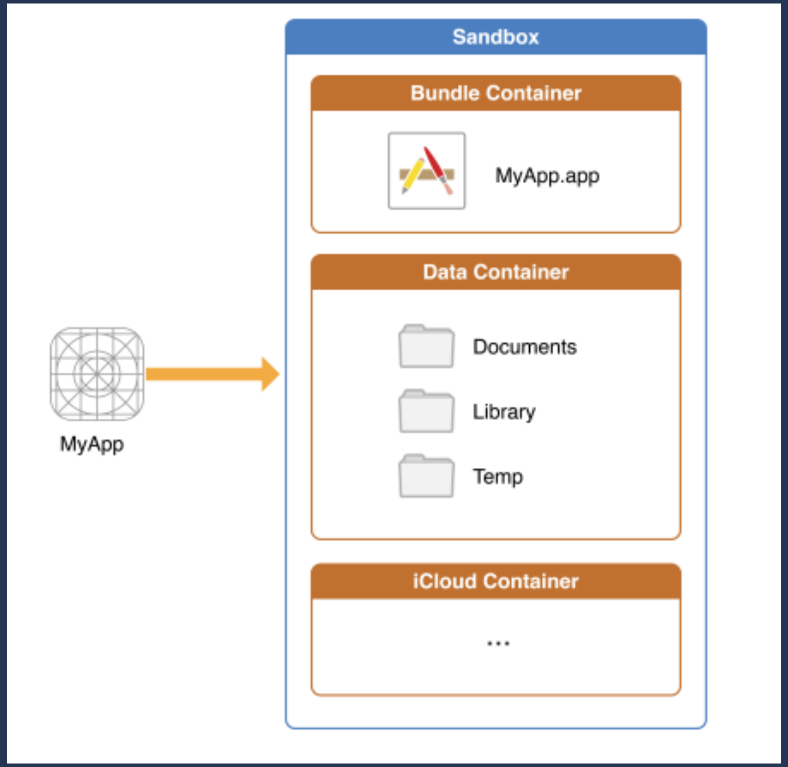
Document generation with appledoc

- Common directives:
  @param [param name]
  @returns
  @see [classname, category name, enum typedef and such]
  @since [version number]
  @exception [exception]
  @warning
  @bug
  @deprecated

Build configurations, schemes & Behaviours

App File System

- Each iOS application has its own sandbox isolated from the rest of the filesystem.
- Your application must stay in its sandbox, and no other application can access your sandbox.



- An App cannot access outside its Sandbox except user’s contacts or music via system Frameworks

App File System

- **App Bundle** . Executable and resources (NIB, images). Signed at installation time. Read only.
- **Documents** . Use this directory to store user-generated content. Backed up by iTunes.
- **Library/Caches** . This directory is where you write data that the application generates during runtime and that you want to persist between runs of the application. NOT Backed up by iTunes.

App File System

- **Library/Preferences** .This directory is where any preferences are stored and where the Settings application looks for application preferences. Handled by NSUserDefaults. Backed up by iTunes.
- **tmp/** . This directory is where you write data that you will use temporarily during an application's runtime. The operating system may purge files in this directory when your application is not running. NOT Backed up by iTunes

Sample code

- Retrieving a text file from the bundle.

```
-(NSString *)textFromBundle {

    NSString *filePath = [[NSBundle mainBundle] pathForResource:@"text" ofType:@"txt"];
    NSString *testString = [NSString stringWithContentsOfFile:filePath encoding:NSUTF8StringEncoding error:nil];

    return testString;
}
```

Sample code

- Retrieving a text file from the Sandbox’s Documents folder.

```
-(NSString *)itemFromDocumentsFolder
{
    NSArray *documentDirectories = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);

    NSString *documentDirectory = [documentDirectories firstObject];

    NSString *testString = [documentDirectory stringByAppendingString:@"/text2.txt"];

    return [NSString stringWithContentsOfFile:testString
                                     encoding:NSUTF8StringEncoding
                                        error:nil];

}
```