Swift Package Manager

or

Let's make dependency manager great, Finally!

Kostiantyn Koval @Agens AS

- Swift from day 1
- Swift Hight Performance
- Swift Package Manager
- @KostiaKoval


pod install

How it used to be 🤔 ...

- Static libraries `lib.a`
- Copied source
- Subproject + Workspace
- Submodules

The problems 😢

- Closed code / Precompiled
- Hard to discover
- No versioning
- Duplicate Symbols

```
a->c, b->c

App ->(a, b) Duplicate 'c'
```

CocoaPods - *Easy*

- Centralized
- Podspec
- Static Libs + Frameworks
- Modifies Xcode projects + workspace

  Res: Workspace

Carthage - Simple

- Decentralized
- No Podspec
- Git + Xcode
- Frameworks

  Res: Frameworks

Happy

Problems

- Not integrated with 🍏
- Xcode private API and Format
- Custom scripts
- Limited

SwiftPM

Why it was built?

~~iOS~~ Linux

SwiftPM

Cross platform, Convention approach, Decentralized

is a tool to automate the process of downloading, compiling, and linking dependencies.
– Swift .org

SwiftPM

Fetch

`swift package`

Build

`swift build`

Test

`swift test`

"swift package"

```
clean                   Delete build artifacts
describe                Describe the current package
dump-package            Print parsed Package.swift as JSON
fetch                   Fetch package dependencies
generate-xcodeproj      Generates an Xcode project
init                    Initialize a new package
reset                   Reset the complete cache/build directory
show-dependencies       Print the resolved dependency graph
update                  Update package dependencies
```

What is a `Swift Package`?

What is a Swift Package ?

Package.swift

```
import PackageDescription

let package = Package(
    name: "Empty"
)
```

What is a Swift Package ?

- Swift, C, C++, Objective-C, Objective-C++

  One language per module

- library (static, dynamic), executable, system-module

Package Convention

Source in -	`/Sources`
Tests in -	`/Tests`
Executable -	`main.swift`
Lib -	`SomeLib.swift`
C headers -	`/include/Baz.h`

Package

```
//executable
.
├── Package.swift
├── Sources
    └── main.swift
```

```
lib with tests
.
├── Package.swift
├── Sources
│   └── Lib.swift
└── Tests
    ├── LibTests
    │   └── LibTests.swift
    └── LinuxMain.swift
```

Package

```
// 2 Modules
.
├── Package.swift
└── Sources
    ├── A
    │   └── A.swift
    └── B
        └── someCode.swift
```

```
system-module
.
├── Package.swift
└── module.modulemap
```

system-module

```
// module.modulemap

module Clibgit [system] {
    header "/usr/local/include/git2.h"
    link "git2"
    export *
}
```


ship IT

Package + Git = 🎉

- Step N1:

`git init, commit, tag 0.1.0, push`

Package + Git = 🎉

- Step N2:

`There is no Step N2`

😀

Package + Git = 🎉

```
let package = Package(
    name: "Empty"
    dependencies: [
        .Package(url: "https://github.com/MyAwesomePackage", majorVersion: 0),
    ]
)
```

semver, semver, semver -> Semver.org

Package Dependency

```
let package = Package(
    name: "Empty"
    dependencies: [
        .Package(url: "https://github.com/MyAwesomePackage", majorVersion: 0),
        .Package(url: "https://github.com/MyAwesomePackage", majorVersion: 1, minor: 4),
        .Package(url: "ssh://git@example.com/Greeter.git", versions: Version(1,0,0)..<Version(2,0,0)),
        .Package(url: "../StringExtensions", "1.0.0-alpha+001"),
        .Package(url: "../Package", version: Version(0, 1, 0),
        .Package(url: "../AwesomePackage",
            version: Version(0, 1, 0, prereleaseIdentifiers: ["alpha"], buildMetadataIdentifier: "001"),
    ]
)
```

Package Power

```
Package(
    name: String,
    dependencies: [Package.Dependency] = [],
    targets: [Target] = [],
    exclude: [String] = []
    pkgConfig: String? = nil,
    providers: [SystemPackageProvider]? = nil,
)
```

Package Targets

3 Targets: Core, Network, Login

```
.
├── Package.swift
└── Sources
    ├── Core
    │   └── core.swift
    ├── Login
    │   └── loginAPI.swift
    └── Network
        └── coreNetwork.swift
```

Package Targets

3 Targets: Core, Network, Login

```
let package = Package(
    name: "App",
    targets: [
        Target(name: "Login", dependencies: ["Core", "Network"]),
    ]
)
```

Package exclude

```
let package = Package(
    name: "Lib",
    exclude: ["Sources/mocJSON",
              "Sources/LibAReadme.md",
              "Tests/FooTests/images"]
)
```

Package pkg-config

```
// module.modulemap
module Clibgit [system] {
    header "/usr/local/include/git2.h"
    link "git2"
    export *
}

swift build -Xcc -I.. -Xlinker -L/usr/local/lib/
```

Package pkg-config

```
swift build -Xcc -I.. -Xlinker -L/usr/local/lib/
```

```
let package = Package(
    name: "Clibgit",
    pkgConfig: "libgit2"
)
```

Package pkg-config

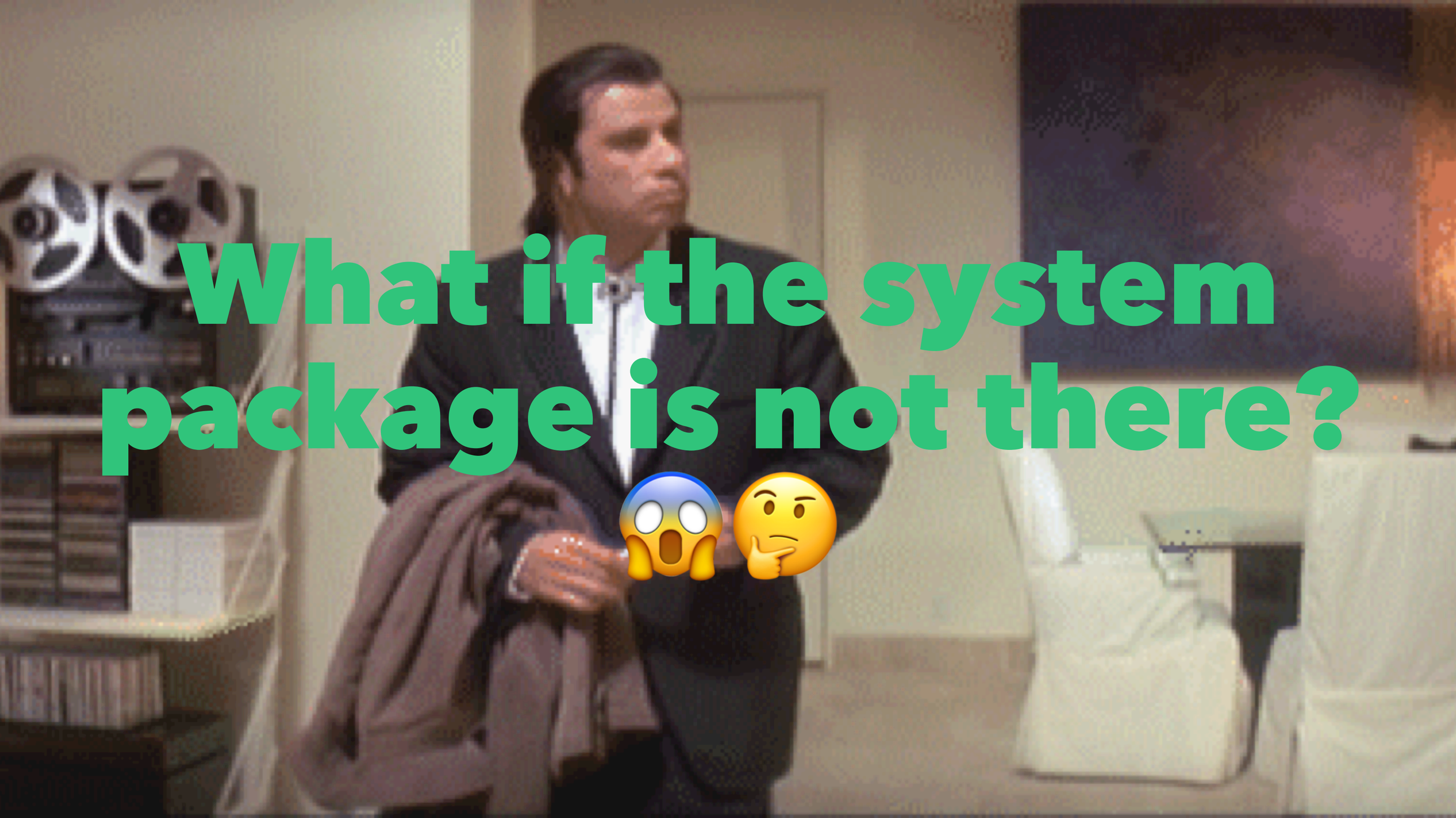
```
swift build
```

```
let package = Package(
    name: "Clibgit",
    pkgConfig: "libgit2"
)
```

Package pkg-config

```
//libgit2.pc file
...

Cflags: -I${includedir}/libgit2
Libs: -L${libdir} -llibgit2
```


What if the system package is not there?
😱🤔

Package Providers

```
let package = Package(
    name: "Clibgit",
    pkgConfig: "libgit2",
    providers: [
        .Brew("libgit2"),
        .Apt("libgit2")
    ]
)
```

Coming Soon ...
Product Definitions

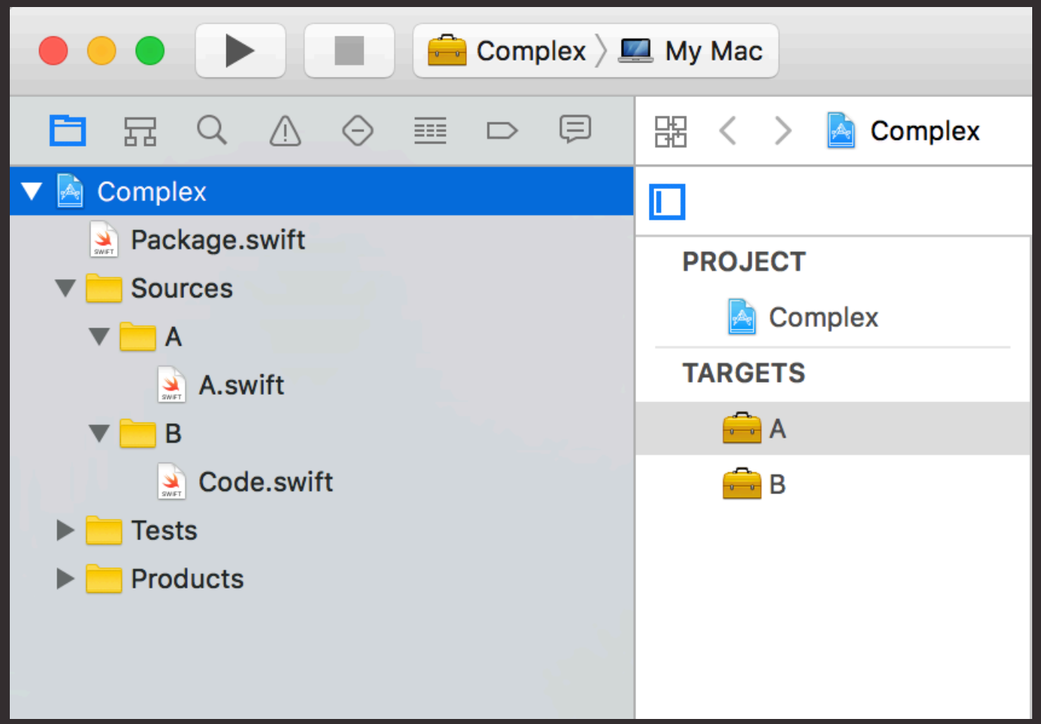
```
let package = Package(
    name: "MyServer",
    ...
    products: [
        .Library(name: "ClientLib", type: .static, targets: ["ClientAPI"]),
        .Library(name: "ServerLib", type: .dynamic, targets: ["ServerAPI"]),
        .Executable(name: "myserver", targets: ["ServerDaemon"]),
    ]
)
```

"*Package Manager Product Definitions*"

Xcode Integration

```
swift package generate-xcodeproj
```

```
.
├── Package.swift
└── Sources
    ├── A
    │   └── A.swift
    └── B
        └── Code.swift
```

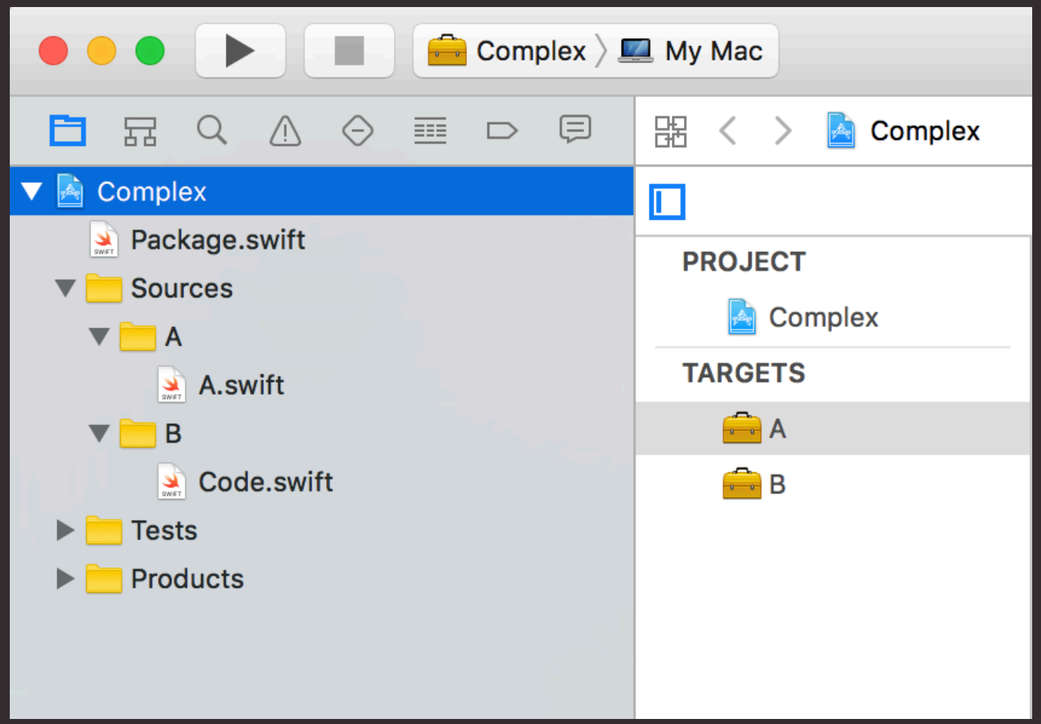


Xcode Integration

```
swift package generate-xcodeproj

--xcconfig-overrides
Config.xcconfig
```

```
.
├── Package.swift
└── Sources
    ├── A
    │   └── A.swift
    └── B
        └── Code.swift
```

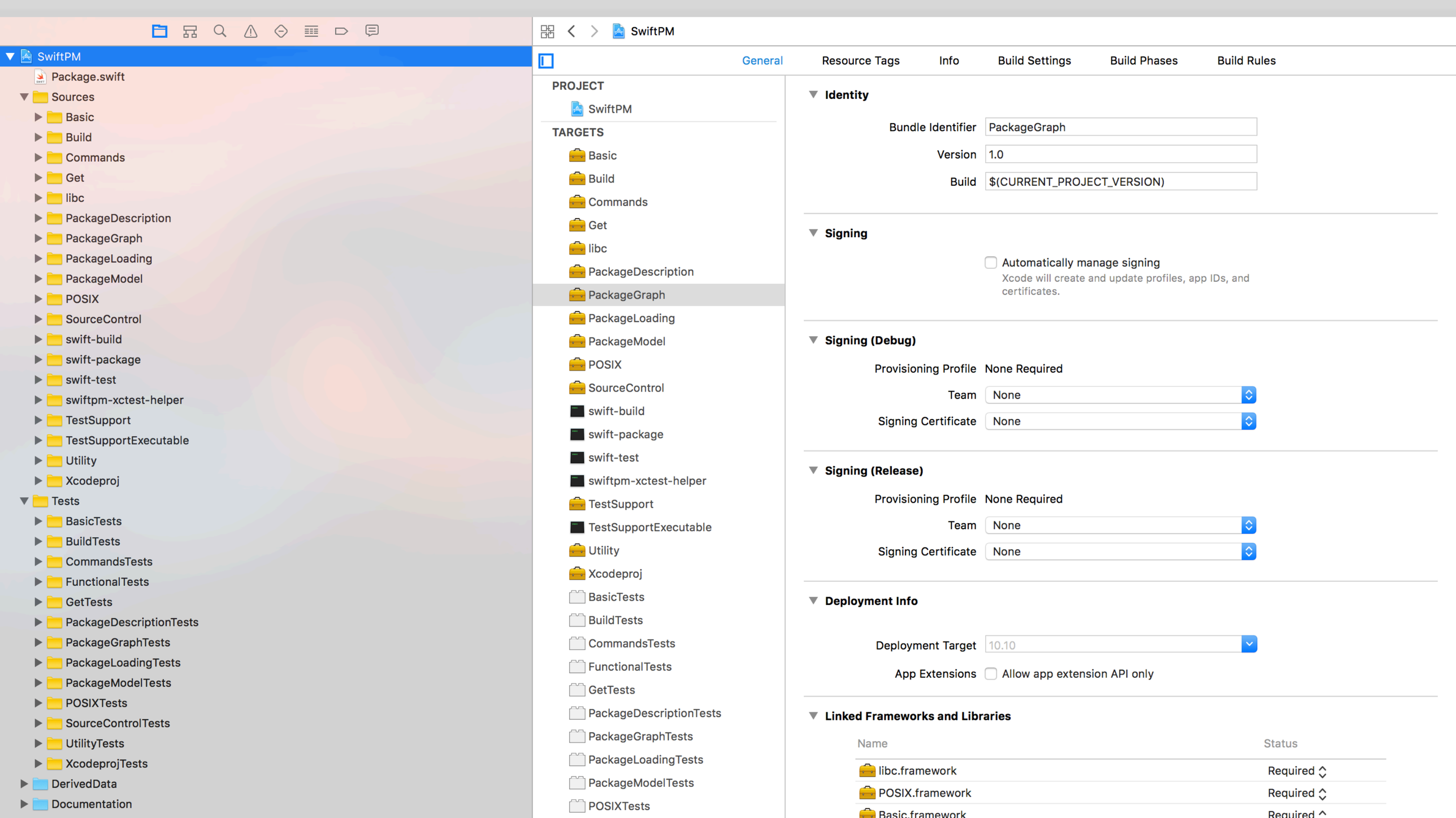


But is it ready for Production and big Projects?

SwiftPM uses SwiftPM to "build and test" SwiftPM

SwiftPM

- 19 Modules
- 31 Targets
- 13 Test Modules
- A lot of Code an Tests


SwiftPM
Package.swift
Sources
Basic
Build
Commands
Get
libc
PackageDescription
PackageGraph
PackageLoading
PackageModel
POSIX
SourceControl
swift-build
swift-package
swift-test
swiftpm-xctest-helper
TestSupport
TestSupportExecutable
Utility
Xcodeproj
Tests
BasicTests
BuildTests
CommandsTests
FunctionalTests
GetTests
PackageDescriptionTests
PackageGraphTests
PackageLoadingTests
PackageModelTests
POSIXTests
SourceControlTests
UtilityTests
XcodeprojTests
DerivedData
Documentation
SwiftPM
General
Resource Tags
Info
Build Settings
Build Phases
Build Rules
PROJECT
SwiftPM
TARGETS
Basic
Build
Commands
Get
libc
PackageDescription
PackageGraph
PackageLoading
PackageModel
POSIX
SourceControl
swift-build
swift-package
swift-test
swiftpm-xctest-helper
TestSupport
TestSupportExecutable
Utility
Xcodeproj
BasicTests
BuildTests
CommandsTests
FunctionalTests
GetTests
PackageDescriptionTests
PackageGraphTests
PackageLoadingTests
PackageModelTests
POSIXTests
Identity
Bundle Identifier
PackageGraph
Version
1.0
Build
$(CURRENT_PROJECT_VERSION)
Signing
Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.
Signing (Debug)
Provisioning Profile
None Required
Team
None
Signing Certificate
None
Signing (Release)
Provisioning Profile
None Required
Team
None
Signing Certificate
None
Deployment Info
Deployment Target
10.10
App Extensions
Allow app extension API only
Linked Frameworks and Libraries
Name
Status
libc.framework
Required
POSIX.framework
Required
Basic.framework
Required

It does handle the Scale

The iOS 😔

Does the SwiftPM solves the issue?

YES!, kind off, it will. 😄

- Painless config
- Right Xcode project integration
- Full Build & Tests control
- Open Source
- Many, many more ...

swift build

The new -

Thanks

@KostiaKoval