Building Faster in Xcode

Session 408

David Owens, Xcode Engineer
Jordan Rose, Swift Engineer
Building Faster in Xcode
# Building Faster in Xcode

| Increasing Build Efficiency | Reducing the Work on Rebuilds |
# Building Faster in Xcode

## Increasing Build Efficiency
- Parallelizing your build process
- Measuring your build time

## Reducing the Work on Rebuilds
- Declaring script inputs and outputs
# Building Faster in Xcode

## Increasing Build Efficiency
- Parallelizing your build process
- Measuring your build time
- Dealing with complex expressions

## Reducing the Work on Rebuilds
- Declaring script inputs and outputs
- Understanding dependencies in Swift
- Limiting your Objective-C/Swift interface
Parallelizing Your Build

Increasing build efficiency
Xcode’s Targets and Dependencies
Xcode’s Targets and Dependencies

Target specifies a product to build

- iOS App
- Framework
- Unit Tests
Xcode’s Targets and Dependencies

Target specifies a product to build
- iOS App
- Framework
- Unit Tests

Target dependency requires another target
- Explicit via Target Dependencies
- Implicit via Link Binary with Libraries
Game Dependency Graph

Tests

Game

Shaders

Utilities

Physics
Game Dependency Graph

List of all targets to build
Game Dependency Graph

List of all targets to build

Dependency between targets

- Game
- Shaders
- Physics
- Utilities
- Tests
Game Dependency Graph

List of all targets to build

Dependency between targets

Build order can be derived

- Physics
- Utilities
- Shaders
- Game
- Tests
Serialized Build Timeline

- Physics
- Utilities
- Shaders
- Game
- Tests
Parallelized Build Timeline
Parallelized Build Timeline

Amount of work did not change
Parallelized Build Timeline

Amount of work did not change

Time to build decreased
Parallelized Build Timeline

Amount of work did not change
Time to build decreased
Parallelized Build Timeline

- Amount of work did not change
- Time to build decreased
- Increased hardware utilization
Extreme Parallelization
How do we get there?
Project Configuration

- Physics
- Utilities
- Game
- Tests
- Shaders

Diagram:
- Tests
- Game
- Shaders
- Physics
- Utilities
Project Configuration

- Physics
- Utilities
- Shaders
- Tests

Diagram:
- Game
  - Shaders
  - Physics
  - Utilities
  - Tests

- Tests
Game Target Dependencies
Game Target Dependencies

- **Game**
  - **Physics**
  - **Utilities**

- **Link Binary With Libraries (2 items)**
  - Physics.framework
  - Utilities.framework
Game Target Dependencies

- Physics
- Utilities
Game Target Dependencies

- Shaders (Shaders)
- Physics
- Utilities
- Game
Game Target Dependencies

- Game
- Shaders
- Physics
- Utilities
Shaders Target Dependencies

Shaders

Utilities
Utilities Target Dependencies

- Physics
- Utilities
Tests Target Dependencies

Game

Shaders

Utilities

Tests
Examine Tests Dependencies

Game

Shaders

Utilities

Tests
Examine Tests Dependencies

UI Tests (Game) → Game

Tests (Shaders) → Shaders

Tests (Utilities) → Utilities
Examine Tests Dependencies

UI Tests (Game) -> Game

Tests (Shaders) -> Shaders

Tests (Utilities) -> Utilities
Reduce Dependency Exposure

Physics

Game

Shaders

Utilities
Reduce Dependency Exposure

- Game
  - Physics
  - Shaders
  - Utilities
Reduce Dependency Exposure
Reduce Dependency Exposure

- Game
- Physics
- Shaders
- Code Gen
- Utilities
Examine Unused Dependencies

Physics

Utilities
Examine Unused Dependencies

Physics

Utilities
Examine Unused Dependencies

- Physics
- Utilities
Target Build Process
Parallelized Target Build Process
Parallelized Target Build Process
Parallelized Target Build Process

Compile sources can start earlier
Parallelized Target Build Process

Compile sources can start earlier

Waits only for what it needs
Parallelized Target Build Process

Compile sources can start earlier

Waits only for what it needs

Must wait for Run Script phases
Run Script Phases
Reducing the work on rebuilds
Allows you to customize your build process for your exact needs.
```bash
# Generate all of the game assets required in our game.

1. *$SRCROOT/Scripts/generate.sh*

2. Show environment variables in build log
3. Run script only when installing

Input Files

- $SRCROOT/Resources/logo.png
- $SRCROOT/Resources/banner.png

Input File Lists

- $SRCROOT/Resources/assets.xcfilelist

Output Files

- $SRCROOT/Resources/logo.png
- $SRCROOT/Resources/banner.png

Output File Lists

- $SRCROOT/Resources/assets.xcfilelist
```
Shell

```bash
# Generate all of the game assets required in our game.
"$SRCROOT/Scripts/generate.sh"
```

- **Show environment variables in build log**
- **Run script only when installing**

```
$SRCROOT/Assets/logo.psd
$SRCROOT/Assets/banner.psd

+ -
Input File Lists
$SRCROOT/Scripts/inputs/assets.xcf/filelist

+ -
Output Files
$SRCROOT/Assets/logo.png
$SRCROOT/Assets/banner.png

+ -
Output File Lists
$SRCROOT/Scripts/outputs/assets.xcf/filelist
```
```bash
#!/bin/sh

# Generate all of the game assets required in our game.
"$SRCROOT/Scripts/generate.sh" -

# Show environment variables in build log
# Run script only when installing

Input Files
$SRCROOT/Assets/logo.psd
$SRCROOT/Assets/banner.psd

+ --
Input File Lists
$SRCROOT/Scripts/inputs/assets.scfilelist

+ --
Output Files
$SRCROOT/Scripts/outputs/assets.scfilelist

+ --
Output File Lists
$SRCROOT/Scripts/outputs/assets.scfilelist
```
Generate Game Assets

Shell

```bash
# Generate all of the game assets required in our game.
"${SRCROOT}/Scripts/generate.sh" -a
```

Input Files

- $(SRCROOT)/Assets/logo.psd
- $(SRCROOT)/Assets/banner.psd

Output Files

- $(SRCROOT)/Scripts/outputs/assets.scfile.txt
- $(SRCROOT)/Outputs/assets.logo.png
- $(SRCROOT)/Outputs/assets.banner.png

Output File List

- $(SRCROOT)/Scripts/Outputs/assets.scfile.txt
Generate all of the game assets required in our game.

Show environment variables in build log

Run script only when installing

Input Files

$SRCROOT/Scripts/logo.psd
$SRCROOT/Assets/logo.psd

Input File Lists

$SRCROOT/Scripts/inputs/assets.xcfilist

Output Files

$SRCROOT/Scripts/logo.png
$SRCROOT/Assets/logo.png

Output File Lists

$SRCROOT/Scripts/outputs/assets.xcfilist
File Lists

# assets.xcfilelist
# Game

# Asset masters
$(SRCROOT)/Assets/game.psd
$(SRCROOT)/Assets/player.psd
$(SRCROOT)/Assets/hud.psd
# assets.xcfilelist
# Game

# Asset masters
$(SRCROOT)/Assets/game.psd
$(SRCROOT)/Assets/player.psd
$(SRCROOT)/Assets/hud.psd
File Lists

Newline separated

Support for build setting variables

# assets.xcfilelist
# Game

# Asset masters
$(SRCROOT)/Assets/game.psd
$(SRCROOT)/Assets/player.psd
$(SRCROOT)/Assets/hud.psd
File Lists

Newline separated

Support for build setting variables

Cannot be generated during the build

# assets.xcfilelist
# Game

# Asset masters
$(SRCROOT)/Assets/game.psd
$(SRCROOT)/Assets/player.psd
$(SRCROOT)/Assets/hud.psd
# Generate all of the game assets required in our game.

```bash
1
2  "$SRCROOT/Scripts/generate.sh"-
3
```

- Show environment variables in build log
- Run script only when installing
- Input Files
  - $SRCROOT/Assets/logo.png
  - $SRCROOT/Assets/banner.png
- Output Files
  - $DERIVED_FILE_DIR/Assets/logo.png
  - $DERIVED_FILE_DIR/Assets/banner.png
- Output File Lists
  - $SRCROOT/Scripts/Outputs/assets.xcfilelist
Shell

```
# Generate all of the game assets required in our game.
#SRCROOT/Scripts/generate.sh -
```

**Run script only when installing**

**Input Files**

- `$(SRCROOT)/Assets/logo.psd`
- `$(SRCROOT)/Assets/banner.psd`

**Output Files**

- `$(DERIVED_FILE_DIR)/Assets/logo.png`
- `$(DERIVED_FILE_DIR)/Assets/banner.png`

- `$(SRCROOT)/Scripts/inputs/assets.scfilelist`
# Generate all of the game assets required in our game.

```sh
#SRCROOT/Scripts/generate.sh`

Show environment variables in build log
Run script only when installing

Input Files
- $SRCROOT/Assets/logo.psd
- $SRCROOT/Assets/banner.psd

Input File Lists
- $SRCROOT/Scripts/inputs/assets.xcfilelist

Output Files
- $SRCROOT/Assets/logo.png
- $SRCROOT/Assets/banner.png

Output File Lists
- $SRCROOT/Scripts/outputs/assets.xcfilelist
```bash
# Generate all of the game assets required in our game.
"$SRCROOT/Scripts/generate.sh"

- Show environment variables in build log
- Run script only when installing

Input Files
- $(SRCROOT)/Assets/logo.png
- $(SRCROOT)/Assets/banner.png

Output File Lists
- $(SRCROOT)/Scripts/Outputs/assets.xcfilelist
- $(SRCROOT)/Scripts/Outputs/assets.png
- $(SRCROOT)/Scripts/Outputs/assets_banner.png
```
When the Run Script Phase is Run

No input files declared

Input files changed

Output files missing
CONFIGURE BUILD PHASES

Run a shell script

You can run a shell script as part of building a target by adding it to the Build Phases pane.

Important: You should provide the inputs and outputs for the shell script because they determine the task execution order when you build targets in parallel. In addition, the inputs and outputs determine if the shell script phase needs to run again for incremental builds. If you don’t specify inputs and outputs, the shell script is run during every build.

1. In the Project navigator, select a project.
2. Choose the target from the Project/Targets pop-up menu or in the Targets section of the second sidebar if it appears.
3. Click Build Phases.
4. Click the Add button (+), then choose “New Run Script Phase” from the pop-up menu.
5. Click the disclosure triangle next to Run Script.
6. In the Shell text field, enter the type of shell, then drag or enter the contents of the shell script below.
   
   If you want to show the environment variables in the build log, select the “Show environment variables in build log” checkbox below the shell script. For additional variables you can use in the shell script, go to Use additional variables.
   
   To write error messages to the build log or trigger a build failure, go to Log errors and warnings and Trigger a build failure.
7. In the Input Files section, enter the files that are inputs to the shell script.
Dependency Cycle Detection

Cycle in dependencies between targets ‘Game’ and ‘Algebra’; building could produce unreliable results.

Build failed 5/29/18, 10:02 PM  1.1 seconds
1 error
TROUBLESHOOT BUILD FAILURES

If your project has a target dependency cycle

The New Build System detects a cycle in dependencies between targets.

The steps of a build often involve high level targets, such as application targets, depending on lower-level targets, such as framework or library targets. The lower level targets are built first, so that the high level targets can link, embed, or otherwise use their products.

If targets have mutual dependencies on each other, a cycle is formed, and it is not correct to build either target first. These mutual dependencies may be direct, or they may be part of a chain that involves other intermediate targets. In either case a build may produce unreliable results.

The New Build System detects dependency cycles and produces a build error explaining the cycle that it detected. If you get this error, you should change the structure of the targets or source code to eliminate the cycle.

Note: To change the build system, go to Choose the build system.

- If the cycle consists of target dependencies only
- If the cycle contains the phrase "Target build order preserved because Parallelize Build is off"
- If the cycle contains script phases which are generating files
- If the cycle is reported only during incremental builds
More on Dependency Cycles

TROUBLESHOOT BUILD FAILURES

 aç If your project has a target dependency cycle

The New Build System detects a cycle in dependencies between targets.

The steps of a build often involve high level targets, such as application targets, depending on lower-level targets, such as framework or library targets. The lower level targets are built first, so that the high level targets can link, embed, or otherwise use their products.

If targets have mutual dependencies on each other, a cycle is formed, and it is not correct to build either target first. These mutual dependencies may be direct, or they may be part of a chain that involves other intermediate targets. In either case a build may produce unreliable results.

The New Build System detects dependency cycles and produces a build error explaining the cycle that it detected. If you get this error, you should change the structure of the targets or source code to eliminate the cycle.

❖ If the cycle consists of target dependencies only

❖ If the cycle contains the phrase "Target build order preserved because Parallelize Build is off"

❖ If the cycle contains script phases which are generating files

❖ If the cycle is reported only during incremental builds
Measuring Build Time
Increasing build efficiency
Build target Algebras
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
Build target Shaders
Project Shaders | Configuration Debug | Destination My Mac | SDK macOS 10.14

/Shader/Particles.metal 0.3 seconds
/Shader/Environment.metal 0.3 seconds
/Shader/Effects.metal 0.3 seconds
/Shader/Shaders.metal 0.3 seconds

Build target CocosX
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14

Build target Physics
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14

Build target Game
Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14

Build Timing Summary
Workspace Game Workspace | Scheme Game | Destination My Mac
PhaseScriptExecution (1 task) 3.036 seconds
CompileAssetsCatalog (1 task) 3.593 seconds
CompileMetaFile (5 tasks) 1.025 seconds
CompileStoryboard (1 task) 1.374 seconds
Ld (5 tasks) 0.340 seconds
CodeSign (3 tasks) 0.186 seconds
CompileC (4 tasks) 0.141 seconds
MetaLink (2 tasks) 0.047 seconds
Touch (5 tasks) 0.040 seconds
LinkStoryboards (1 task) 0.023 seconds

Build succeeded 5/29/18, 12:09 AM 0.4 seconds
No issues
Build Log Filters
Build Log Filters
Prepare build
Workspace Game Workspace | Scheme Game | Destination My Mac
Using new build system
Planning build
Using build description from memory

Build target Algorithms
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
Build target Algorithms
Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14

Build target Shaders
Project Shaders | Configuration Debug | Destination My Mac | SDK macOS 10.14
Compile /Users/ownswi/WWDC 2018/Building Faser in Xcode/Demo App/Before/Shaders/ShadersParticles.metal 0.3 seconds
Compile /Users/ownswi/WWDC 2018/Building Faser in Xcode/Demo App/Before/Shaders/Shaders/Environment.metal 0.3 seconds
Compile /Users/ownswi/WWDC 2018/Building Faser in Xcode/Demo App/Before/Shaders/Shaders/Effects.metal 0.3 seconds
Compile /Users/ownswi/WWDC 2018/Building Faser in Xcode/Demo App/Before/Shaders/Shaders/Shaders.metal 0.3 seconds
Linking Metal All /Users/ownswi/Library/Developer/Xcode/DerivedData/Game_Workspace-dmpmpfljv1jpojcpkimqjwcnl/Build/Products/Debug/Shaders/Shaders.metal 0.1 seconds

Build target Calculus
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14
Build target Physics
Project Physics | Configuration Debug | Destination My Mac | SDK macOS 10.14

Build target Game
Project Game | Configuration Debug | Destination My Mac | SDK macOS 10.14

Build Timing Summary
Workspace Game workspace | Scheme Game | Destination My Mac
Phase/GorupExecution (1 task) 1.038 seconds
CompileGorupCatalog (1 task) 2.563 seconds
CompileGorupFile (5 tasks) 1.635 seconds
CompileGorupHeader (1 task) 1.374 seconds
Ld (5 tasks) 0.301 seconds
CodeGen (3 tasks) 0.186 seconds
CompileC (4 tasks) 0.141 seconds
Merge (2 tasks) 0.047 seconds
Touch (5 tasks) 0.040 seconds
LinkGorupHeader (1 task) 0.023 seconds
Build succeeded 5/18/2019 12:04 AM 9.4 seconds

No issues
Build target Algebra

Build target Utilities

Build target Shaders

Build target Calculus

Build target Physics

Build target Game

Build Timing Summary

Workspace Game Workspace | Scheme Game | Destination My Mac

PhaseScriptExecution (1 task)  5.036 seconds
CompileAssetCatalog (1 task)  2.552 seconds
CompileMetaFile (5 tasks)  1.625 seconds
CompileStoryboard (1 task)  1.374 seconds
Ld (5 tasks)  0.340 seconds
CodeSign (3 tasks)  0.186 seconds
CompileC (4 tasks)  0.141 seconds
MetalLink (2 tasks)  0.047 seconds
Touch (5 tasks)  0.040 seconds
LinkStoryboards (1 task)  0.023 seconds

Build succeeded  5/25/18, 12:08 AM  9.4 seconds
No issues
Build Timing Summary

Workspace Game Workspace | Scheme Game | Destination My Mac

- PhaseScriptExecution (1 task)  5.036 seconds
- CompileAssetCatalog (1 task)  2.552 seconds
- CompileMetalFile (5 tasks)  1.625 seconds
- CompileStoryboard (1 task)  1.374 seconds
- Ld (5 tasks)  0.340 seconds
- CodeSign (3 tasks)  0.186 seconds
- CompileC (4 tasks)  0.141 seconds
- Metallink (2 tasks)  0.047 seconds
- Touch (5 tasks)  0.040 seconds
- LinkStoryboards (1 task)  0.023 seconds
Build Timing Summary

Workspace Game Workspace | Scheme Game | Destination My Mac

PhaseScriptExecution (1 task) 5.036 seconds

- CompileAssetCatalog (1 task) 2.552 seconds
- CompileMetalFile (5 tasks) 1.625 seconds
- CompileStoryboard (1 task) 1.374 seconds
- Ld (5 tasks) 0.340 seconds
- CodeSign (3 tasks) 0.186 seconds
- CompileC (4 tasks) 0.141 seconds
- Metallink (2 tasks) 0.047 seconds
- Touch (5 tasks) 0.040 seconds
- LinkStoryboards (1 task) 0.023 seconds
$ xcodebuild -showBuildTimingSummary

Build Timing Summary

CompileSwiftSources (1 task) | 5.434 seconds
PhaseScriptExecution (1 task) | 5.046 seconds
CompileAssetCatalog (1 task) | 2.788 seconds
CompileStoryboard (1 task) | 1.880 seconds
CompileMetalFile (5 tasks) | 1.735 seconds
CopySwiftLibs (1 task) | 0.740 seconds
Ld (2 tasks) | 0.306 seconds
CodeSign (3 tasks) | 0.177 seconds
CompileC (1 task) | 0.170 seconds
MetalLink (2 tasks) | 0.046 seconds
Ditto (4 tasks) | 0.032 seconds
LinkStoryboards (1 task) | 0.023 seconds
$ xcodebuild -showBuildTimingSummary

Build Timing Summary

CompileSwiftSources (1 task) | 5.434 seconds
PhaseScriptExecution (1 task) | 5.046 seconds
CompileAssetCatalog (1 task) | 2.788 seconds
CompileStoryboard (1 task) | 1.880 seconds
CompileMetalFile (5 tasks) | 1.735 seconds
CopySwiftLibs (1 task) | 0.740 seconds
Ld (2 tasks) | 0.306 seconds
CodeSign (3 tasks) | 0.177 seconds
CompileC (1 task) | 0.170 seconds
MetalLink (2 tasks) | 0.046 seconds
Ditto (4 tasks) | 0.032 seconds
LinkStoryboard (1 task) | 0.023 seconds
Source-Level Improvements

Jordan Rose, Swift Engineer
Remove this build time workaround...
Reducing the work on rebuilds
Turn off Whole Module Mode For Debug Builds
Turn off Whole Module Mode For Debug Builds
Turn off Whole Module Mode For Debug Builds

What’s New in Swift

Behind the Scenes of the Xcode Build Process
<table>
<thead>
<tr>
<th>Increasing Build Efficiency</th>
<th>Reducing the Work on Rebuilds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallelizing your build process</td>
<td>Declaring script inputs and outputs</td>
</tr>
<tr>
<td>Measuring your build time</td>
<td>Understanding dependencies in Swift</td>
</tr>
<tr>
<td>Dealing with complex expressions</td>
<td>Limiting your Objective-C/Swift interface</td>
</tr>
</tbody>
</table>
Dealing with complex expressions
Understanding dependencies in Swift
Limiting your Objective-C/Swift interface
When a build takes a long time, there is often a key piece of information you can add to make it better.
Dealing with Complex Expressions

Increasing build efficiency
Use Explicit Types for Complex Properties

```swift
struct ContrivedExample {
    var bigNumber = [4, 3, 2].reduce(1) {
        soFar, next in
        pow(next, soFar)
    }
}
```
Use Explicit Types for Complex Properties

```swift
struct ContrivedExample {
    var bigNumber: Double = [4, 3, 2].reduce(1) {
        soFar, next in
        pow(next, soFar)
    }
}
```
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        soFar, next in
    }
}
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        soFar, next in
        soFar != nil && next != nil ? soFar! + next! :
        (soFar != nil ? soFar! : (next != nil ? next! : nil))
    }
}
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        soFar, next in
        soFar != nil && next != nil ? soFar! + next! :
        (soFar != nil ? soFar! : (next != nil ? next! : nil))
    }
}
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        soFar, next in
        soFar != nil && next != nil ? soFar! + next! :
            (soFar != nil ? soFar! : (next != nil ? next! : nil))
    }
}
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        (soFar: Int?, next: Int?) -> Int? in
        soFar != nil && next != nil ? soFar! + next! :
            (soFar != nil ? soFar! : (next != nil ? next! : nil))
    }
}
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        soFar, next in
        soFar != nil && next != nil ? soFar! + next! :
            (soFar != nil ? soFar! : (next != nil ? next! : nil))
    }
}
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        soFar, next in
        soFar != nil && next != nil ? soFar! + next! :
            (soFar != nil ? soFar! : (next != nil ? next! : nil))
    }
}
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        soFar, next in
        soFar != nil && next != nil ? soFar! + next! :
        (soFar != nil ? soFar! : (next != nil ? next! : nil))
    }
}
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        soFar, next in
        soFar != nil && next != nil ? soFar! + next! :
            (soFar != nil ? soFar! : (next != nil ? next! : nil))
    }
}
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        soFar, next in
    }
}
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        soFar, next in
        if soFar != nil && next != nil { return soFar! + next! }
        if soFar != nil { return soFar! }
        if next != nil { return next! }
        return nil
    }
}
func sumNonOptional(i: Int?, j: Int?, k: Int?) -> Int? {
    return [i, j, k].reduce(0) {
        soFar, next in
        if let soFar = soFar {
            if let next = next { return soFar + next }
            return soFar
        } else {
            return next
        }
    }
}
Use `AnyObject` Methods and Properties Sparingly

```swift
weak var delegate: AnyObject?

func reportSuccess() {
    delegate?.myOperationDidSucceed(self)
}
```
Use **AnyObject** Methods and Properties Sparingly

```swift
weak var delegate: AnyObject?

func reportSuccess() {
    delegate?.myOperationDidSucceed(self)
}
```
Use `AnyObject` Methods and Properties Sparingly

```swift
weak var delegate: AnyObject?

func reportSuccess() {
    delegate?.myOperationDidSucceed(self)
}
```
weak var delegate: AnyObject?

func reportSuccess() {
    delegate?.myOperationDidSucceed(self)
}

Use **AnyObject** Methods and Properties Sparingly
Use `AnyObject` Methods and Properties Sparingly
Define a protocol instead

```swift
weak var delegate: AnyObject?
func reportSuccess() {
    delegate?.myOperationDidSucceed(self)
}

protocol MyOperationDelegate: class {
    func myOperationDidSucceed(_ operation: MyOperation)
}
```
Use **AnyObject** Methods and Properties Sparingly
Define a protocol instead

```swift
weak var delegate: MyOperationDelegate?

func reportSuccess() {
    delegate?.myOperationDidSucceed(self)
}

protocol MyOperationDelegate: class {
    func myOperationDidSucceed(_ operation: MyOperation)
}
```
Use `AnyObject` Methods and Properties Sparingly
Define a protocol instead

```swift
 weak var delegate: MyOperationDelegate?
 func reportSuccess() {
   delegate?.myOperationDidSucceed(self)
 }

 protocol MyOperationDelegate: class {
   func myOperationDidSucceed(_ operation: MyOperation)
 }
```
Understanding Dependencies in Swift
Reducing the work on rebuilds
Incremental Builds Are File-Based

```swift
struct Point {
    var x, y: Double
}
```
Incremental Builds Are File-Based

```swift
struct Point {
    var x, y: Double
}

let point = Point(x: 3.0, y: 4.0)
let distance = sqrt(point.x * point.x + point.y * point.y)
```
Incremental Builds Are File-Based

```swift
struct Point {
    var x, y: Double
}

let point = Point(x: 3.0, y: 4.0)
let distance = sqrt(point.x * point.x + point.y * point.y)
```
Incremental Builds Are File-Based

```swift
struct Point {
    var x, y: Double

    init(x: Double, y: Double) {
        assert(x.isNormal)
        assert(y.isNormal)
        self.x = x
        self.y = y
    }
}
```

```swift
let point = Point(x: 3.0, y: 4.0)
let distance = sqrt(point.x * point.x + point.y * point.y)
```
struct Point {
    var x, y: Double

    init(x: Double, y: Double) {
        assert(x.isNormal)
        assert(y.isNormal)
        self.x = x
        self.y = y
    }
}

let point = Point(x: 3.0, y: 4.0)
let distance =
    sqrt(point.x * point.x +
    point.y * point.y)
Incremental Builds Are File-Based
Changes in function bodies do not affect other files

```swift
struct Point {
    var x, y: Double
    init(x: Double, y: Double) {
        assert(x.isFinite)
        assert(y.isFinite)
        self.x = x
        self.y = y
    }
}
```

```swift
let point = Point(x: 3.0, y: 4.0)
let distance = sqrt(point.x * point.x + point.y * point.y)
```
Incremental Builds Are File-Based
Changes in function bodies do not affect other files

```swift
struct Point {
    var x, y: Double
    init(x: Double, y: Double) {
        assert(x.isFinite)
        assert(y.isFinite)
        self.x = x
        self.y = y
    }
}

let point = Point(x: 3.0, y: 4.0)
let distance = sqrt(point.x * point.x + point.y * point.y)
```
Incremental Builds Are File-Based
Unrelated changes outside function bodies can still result in rebuilding

```swift
struct Point {
    var x, y: Double

    init(x: Double, y: Double) {
        assert(x.isFinite)
        assert(y.isFinite)
        self.x = x
        self.y = y
    }
}

struct PathSegment {
    var start, end: Point
}

let point = Point(x: 3.0, y: 4.0)
let distance = sqrt(point.x * point.x + point.y * point.y)
```
Incremental Builds Are File-Based
Unrelated changes outside function bodies can still result in rebuilding

```swift
struct Point {
    var x, y: Double

    init(x: Double, y: Double) {
        assert(x.isFinite)
        assert(y.isFinite)
        self.x = x
        self.y = y
    }
}

struct PathSegment {
    var start, end: Point
}

let point = Point(x: 3.0, y: 4.0)
let distance = sqrt(point.x * point.x + point.y * point.y)
```
Dependencies Within a Target Are Per-File

- GameView.swift
- SyncManager.swift
- MenuController.swift
- GameAI.swift
- PlayerState.swift
Dependencies Within a Target Are Per-File

- GameView.swift
- SyncManager.swift
- MenuController.swift
- GameAI.swift
- PlayerState.swift
Dependencies Within a Target Are Per-File

- GameView.swift
- SyncManager.swift
- MenuController.swift
- GameAI.swift
- PlayerState.swift
Dependencies Within a Target Are Per-File

GameView.swift  SyncManager.swift  MenuController.swift

GameAI.swift  PlayerState.swift
Dependencies Within a Target Are Per-File

- GameView.swift
- SyncManager.swift
- MenuController.swift
- GameAI.swift
- PlayerState.swift
Cross-Target Dependencies Are Coarse-Grained

MenuController.swift

GameView.swift  SyncManager.swift  MenuController.swift

PlayerState.swift

GameAI.swift  PlayerState.swift
Cross-Target Dependencies Are Coarse-Grained

GameView.swift  SyncManager.swift  MenuController.swift

GameAI.swift  PlayerState.swift
Cross-Target Dependencies Are Coarse-Grained

- GameView.swift
- SyncManager.swift
- MenuController.swift
- GameAI.swift
- PlayerState.swift
Cross-Target Dependencies Are Coarse-Grained

GameView.swift  SyncManager.swift  MenuController.swift

GameAI.swift  PlayerState.swift
Cross-Target Dependencies Are Coarse-Grained

GameView.swift  SyncManager.swift  MenuController.swift

GameAI.swift  PlayerState.swift
Swift Dependency Rules
Swift Dependency Rules

Compiler must be conservative
Swift Dependency Rules

Compiler must be conservative

Changes in function bodies do not affect the file’s interface
Swift Dependency Rules

Compiler must be conservative

Changes in function bodies do not affect the file’s interface

Dependencies within a module are per-file
Swift Dependency Rules

Compiler must be conservative

Changes in function bodies do not affect the file’s interface

Dependencies within a module are per-file

Dependencies across targets are for the whole target
Limiting Your Objective-C/Swift Interface
Reducing the work on rebuilds
Mixed-Source App Targets
Mixed-Source App Targets
Mixed-Source App Targets

Bridging header
Mixed-Source App Targets

Bridging header

SWIFT

SWIFT

SWIFT

SWIFT
Mixed-Source App Targets

Bridging header

Generated header
Mixed-Source App Targets

Bridging header

Generated header

h

h

h

h

h

h

m

m

SWIFT

SWIFT

SWIFT

SWIFT

m
Mixed-Source App Targets

Bridging header

 Generated header
Mixed-Source App Targets
Mixed-Source App Targets

Bridging header

Generated header
class MainViewController: UIViewController {
    @IBOutlet var statusField: UITextField!
    @IBAction func close(_ sender: Any?) { ... }
Keep Your Generated Header Minimal
Use private when possible

class MainViewController: UIViewController {
    @IBOutlet var statusField: UITextField!
    @IBAction func close(_ sender: Any?) { … } 
}

@property IBOutlet UITextField *statusField;
-(IBAction) close:(id)sender;
Keep Your Generated Header Minimal
Use `private` when possible

class MainViewController: UIViewController {
    @IBOutlet private var statusField: UITextField!
    @IBAction private func close(_ sender: Any?) {
    }
}

@property IBOutlet UITextField *statusField;
-(IBAction)close:(id)sender;
Keep Your Generated Header Minimal
Use private when possible

class MainViewController: UIViewController {
    @IBOutlet private var statusField: UITextField!
    @IBAction private func close(_ sender: Any?) { ... }
}
Keep Your Generated Header Minimal
Use private when possible

class MainViewController: UIViewController {
    @IBOutlet private var statusField: UITextField!
    @IBAction private func close(_ sender: Any?) { ... }
}

Swift

Header Minimalization

- Use `private` to keep your generated headers small and less cluttered.
- Avoid unnecessary class and instance variables.
- Minimize the scope of local variables to reduce code complexity.
Keep Your Generated Header Minimal
Use private when possible

@objc func keyboardWillShow(_: Notification) {
    // Important keyboard setup code here.
}

// …
NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillShow(_:)), …)
Keep Your Generated Header Minimal
Use private when possible

@objc private func keyboardWillShow(_: Notification) {
    // Important keyboard setup code here.
}

// …
NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillShow(_:)), …)
Keep Your Generated Header Minimal
Use block-based APIs

```swift
self.observer = NotificationCenter.default.addObserver(
    forName: UIKeyboardWillShow, object: nil, queue: nil) {
    // Important keyboard setup code here.
}
```
Keep Your Generated Header Minimal

Migrate to Swift 4
Keep Your Generated Header Minimal
Turn off “Swift 3 @objc Inference”
Keep Your Generated Header Minimal
Turn off “Swift 3 @objc Inference”
Keep Your Bridging Header Minimal

// Use this file to import your target's public headers that
// you would like to expose to Swift.

#import "MyBackendAPI.h"
#import "MyViewController.h"
#import "MyUserDefaultsHelpers.h"
// Use this file to import your target's public headers that
// you would like to expose to Swift.

#import "MyBackendAPI.h"
#import "MyViewController.h"
#import "MyUserDefaultsHelpers.h"
Keep Your Bridging Header Minimal

```cpp
#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"

@interface MyViewController: UIViewController
@property (nonnull) MyNetworkManager *networkManager;
// ...
@end
```
Keep Your Bridging Header Minimal

#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"

@interface MyViewController: UIViewController
@property (nonnull) MyNetworkManager *networkManager;
// ...
@end
Keep Your Bridging Header Minimal

#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"

@interface MyViewController: UIViewController
@property (nonnull) MyNetworkManager *networkManager;
// ...
@end
Keep Your Bridging Header Minimal
Use categories to break up your interface

#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"
@interface MyViewController: UIViewController
@property (nonnull) MyNetworkManager *networkManager;
// ...
@end
Keep Your Bridging Header Minimal
Use categories to break up your interface

```objective-c
#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"
@interface MyViewController: UIViewController
@property (nonnull) MyNetworkManager *networkManager;
// ...
@end
```

```objective-c
#import "MyViewController.h"
@interface MyViewController ()
@end
```

```objective-c
#import "MyViewController.h"
@interface MyViewController+Internal.h
@end
```
Keep Your Bridging Header Minimal
Use categories to break up your interface

#import <UIKit/UIKit.h>
#import "MyNetworkManager.h"
@interface MyViewController: UIViewController
@property (nonnull) MyNetworkManager *networkManager;
// ...
@end

#import "MyViewController.h"
@interface MyViewController ()
@end

#import "MyViewContoller+Internal.h"
Keep Your Bridging Header Minimal

Use categories to break up your interface

```c
#import <UIKit/UIKit.h>
@interface MyViewController: UIViewController
// ...
@end

#import "MyNetworkManager.h"
#import "MyViewController.h"
@interface MyViewController ()
@property (nonnull) MyNetworkManager *networkManager;
@end
```

```c
#import <UIKit/UIKit.h>
@interface MyViewController: UIViewController
// ...
@end

#import "MyNetworkManager.h"
#import "MyViewController.h"
@interface MyViewController ()
@property (nonnull) MyNetworkManager *networkManager;
@end
```
Keep Your Bridging Header Minimal

Use categories to break up your interface

```cpp
#import <UIKit/UIKit.h>
@interface MyViewController: UIViewController
// …
@end
```

```cpp
#import "MyNetworkManager.h"
#import "MyViewController.h"
@interface MyViewController :
@property (nonnull) MyNetworkManager *networkManager;
@end
```
Less Content, Fewer Changes, Faster Builds

Bridging header

Generated header
Less Content, Fewer Changes, Faster Builds

Bridging header

Generated header
Less Content, Fewer Changes, Faster Builds

Bridging header

Generated header
Summary
## Building Faster in Xcode

### Increasing Build Efficiency
- Parallelizing your build process
- Measuring your build time
- Dealing with complex expressions

### Reducing the Work on Rebuilds
- Declaring script inputs and outputs
- Understanding dependencies in Swift
- Limiting your Objective-C/Swift interface
More Information

https://developer.apple.com/wwdc18/408

<table>
<thead>
<tr>
<th>Building Your App with Xcode</th>
<th>Technology Lab 8</th>
<th>Thursday 12:00PM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building Your App with Xcode</td>
<td>Technology Lab 8</td>
<td>Friday 3:00PM</td>
</tr>
</tbody>
</table>