Xcode Release Notes

About Xcode 7 beta 6

Supported Configurations

Xcode 7 beta 6 requires a Mac running OS X 10.10.4 or later.

Xcode 7 beta includes SDKs for watchOS 2.0, iOS 9 and OS X version 10.11. To develop apps targeting prior versions of OS X or iOS, see the section “About SDKs and the Simulator” in *What's New in Xcode* available on developer.apple.com or from the Help > What's New in Xcode command when running Xcode.

Xcode 6.x has known compatibility issues when run on OS X El Capitan. It is strongly encouraged that you use Xcode 7 when running on OS X El Capitan.

Installation

To install Xcode 7 beta, double-click the downloaded DMG file, and drag the Xcode-beta.app file to your Applications folder.

From within Xcode, you can launch additional developer tools, such as Instruments and FileMerge, using the Xcode > Open Developer Tool menu. You can keep the additional tools in the Dock for direct access when Xcode is not running.

Installing Xcode on OS X Server

To use Xcode’s continuous integration service with this Xcode beta, you need OS X 10.10.4 or greater with OS X Server 4.1 or greater

Once you have installed OS X, OS X Server and Xcode, point OS X Server to this Xcode beta with these steps:

1. Install the Server app and Xcode
2. Launch Xcode, accept Xcode license agreement and install additional components if necessary
3. Launch the Server app, initialize the server
4. Once Server is initialized, go to the Xcode tab, click on Choose Xcode and select Xcode 7
5. Once Xcode Server is initialized, turn Xcode Service on

Technical Support and Learning Resources

Apple offers a number of resources where you can get Xcode development support:

http://developer.apple.com: The Apple Developer website is the best source for up-to-date technical documentation on Xcode, iOS, and OS X.

http://developer.apple.com/xcode: The Xcode home page on the Apple Developer website provides information about acquiring the latest version of Xcode.

http://devforums.apple.com: The Apple Developer Forums are a good place to interact with fellow developers and Apple engineers, in a moderated web forum that also offers email notifications. The Developer Forums also feature a dedicated topic for Xcode developer previews.

Use http://bugreport.apple.com to report issues to Apple. Include detailed information of the issue, including the system and developer tools version information, and any relevant crash logs or console messages.

Deprecations and Removal Notices

- OS X 10.11 is the last major release of OS X that will support the previously deprecated garbage collection runtime. Applications or features that depend upon garbage collection may not function properly or will not launch when the runtime is removed. Developers should use Automatic Reference Counting (ARC) or manual retain/release for memory management. (20589595)

Xcode 7 beta 6 — New Feature Highlights

Platform Support

- Xcode 7 beta 6 includes SDKs for watchOS 2.0, OS X version 10.11, and iOS 9.

Swift Language

- **Error Handling.** Throw, catch, manage, and handle errors in Swift. Interoperate seamlessly with NSError.
- **Availability.** Adopt new APIs while still deploying back to older OS versions, with compile-time errors to catch situations when you've used API that isn't available on the deployment target.
- **Protocol extensions.** Add methods and properties to any class that conforms to a protocol. Re-use more of your code.
- **Testability.** Write tests of Swift 2.0 frameworks and apps with access to all your public and internal routines.
- **Swift 1.2 to 2.0 Migrator.** Helps you upgrade your existing Swift source code to take advantage of Swift 2.0.

Objective-C Updates

- **Generics.** Specify type information for collections, simplifying the code you write.
- **Nullability.** Indicate in Objective-C source when to expect nil or non-nil results.

Playgrounds

- **Rich-text comments.** Explain what is going on in the Swift code with a markdown-like syntax.
- **Inline results.** Show your code's results directly below the code that produces them.
- **Resources.** Add resources like images to your playground using the project navigator.
- **Auxiliary Sources.** Keep additional support code separate from the playground itself.
- **Pages.** Bundle related concepts together with multiple, targeted to structure a playground.

App Thinning

Three complementary technologies designed to deliver optimized installation by the App Store.

- **Bitcode.** Archive for upload to the App Store in an intermediate LLVM binary representation that the store can then optimize into the 64 or 32-bit executable to be delivered to customers.

- **Slicing.** Artwork incorporated into the Asset Catalog and tagged for platform needs allows the App Store to deliver only what is needed for installation.

- **On Demand Resources.** Host additional content for your app on the iTunes App Store repository, allowing the app to fetch resources as needed using asynchronous download and installation.

Debugging

- **Energy Gauge for iOS.** Track energy usage with iOS 9 on a per-process basis.

- **Address Sanitizer.** Build instrumented Objective-C and C code to trap the sources of memory corruption problems.

Testing

- **User interface testing and recording.** Test applications at the user interface surface with elements, queries, and simulated events. The UI recording feature enables capture of UI actions into source to facilitate creating tests.

- **Code coverage.** Provides report information to evaluate a test suite for completeness.

Free Provisioning

- **Develop on your own device.** Streamlined mechanism to provision and install development project on physical devices for testing and evaluation.

Crash Logs

- **Test Flight and Crash Reports.** Enhanced to allow using crash data from OS X apps as well as watchOS and iOS apps.

New in Xcode 7 beta 6 — IDE

UI Testing

- XCTest offers API to automate complex controls such as date pickers. (20577276)

Resolved in Xcode 7 beta 6 — IDE

UI Testing

- Apps with names that contain parenthesis can be exported from the Archives organizer. (22142125)

watchOS App Installation

- The `STRIP_BITCODE_FROM_COPIED_FILES` build setting is no longer needed to install Watch apps written with Swift code from an iPhone to a Watch. (22139044)

Xcode Source Control Management

- Xcode no longer crashes when editing a document while it is being unlocked. (21528657)

Known Issues in Xcode 7 beta 6 — IDE

General

- Items in Xcode's find navigator scope picker outline view are invisible on OS X 10.11.

  Workaround: Moving the Xcode window will cause the outline view to redraw. (20275994)

- Creating new projects on OS X El Capitan requires developer beta 7 or later. (22263243)

Simulator

- Background upload/download using `NSURLSession` completes, but notifications do not badge the app to show completion when the app is in the background.

  Workaround: Testing on device works as expected. (16532261)

- Playing video to a simulated external display does not function correctly with iPad simulators.

  Workaround: Use an iPhone simulator. (17979778)

- Video playback is often incorrectly cropped or translated when playing back on a TVOut display in Simulator.

  Workaround: Test video playback on a physical device. (19394195)

- Playing back video to a 1080p external display in Simulator renders at a lower resolution.

  Workaround: Play video at a lower resolution or test 1080p playback with a physical device. (18296724)

Accounts

- Xcode does not support two-factor authentication. Developers should continue to authenticate with username and password. (19581582)

- Existing source control accounts may appear with empty passwords after launching Xcode 7.

  Workaround: Re-enter the password. (21252258)

Archiving Xcode Projects

- If your iOS app embeds a watchOS app and you have a framework for both iOS and watchOS embedded in each app, your project will fail to build when you perform the Archive action.

  Workaround: Use a different product name (using the PRODUCT_NAME build setting) for the watchOS version of your framework. (22183332)

Localization

- During XLIFF export or import, `NSLocalizedString` macro issues or empty strings files may result in an error, "The data couldn't be read because it isn't in the correct format."

    Workaround: Remove empty strings files from your project, or use the following command to find NSLocalizedString macro issues in your project.

```
find <project directory here> -name "*.m" -exec xcrun extractLocStrings {} \;
```

  (21101899)

Playgrounds

- Opening a playground that was previously displaying the version editor may present an alert stating, "The file "MyPlayground.playground" couldn't be opened because you don't have permission to view it."

    Workaround: Dismiss the alert dialog and show the standard editor (Command-Return). (20623808)

- Playgrounds live views may not show images when using the timeline slider. (21261310)

- Comparison and Blame view in the Versions editor may not produce results for Playgrounds. (21879794)

- Playgrounds created by previous versions of Xcode may not be upgradable to Swift 2. (20902099)

    Workaround: Select `Editor > Upgrade Playground` to upgrade the playground before attempting to it convert to Swift 2.

Devices

- Restoring your device while Xcode is open may cause your device to appear as unavailable or locked to Xcode after the restore is complete.

    Workaround: Re-attaching your device after a restore fixes this issue. (21344895)

UI Testing

- UI testing cannot identify elements using information from their accessibility title element.

    Workaround: Set an accessibility identifier instead. (20409319)

- UI testing cannot record or interact with popovers on OS X. (21162677)

- UI tests do not support key modifiers on iOS. (20185910, 21033224)

- When recording a UI Test for an OS X app, Xcode Helper, not Xcode, asks to use the Accessibility APIs. This app uses the Accessibility APIs when running UI Tests. (21211897)

Xcode Server

- When Runtime Sanitization is enabled for an Xcode Server bot, detailed information about any crash triggered by sanitization checks is made available in the raw build log for the integration. (21047341)
- Continuous integration bot email notifications may not send by default on some networks.

  Workaround: Configure a custom SMTP relay server using:

  ```
  sudo xcrun xcscontrol --configure-email-transport server
  ```

  (21189162)

On-Demand Resources

- While debugging On Demand Resources applications from Xcode, the progress of tag requests may reset to zero when moving the application to or from the foreground or when changing the loadingPriority of an NSBundleResourceRequest. (21882271)

OS X Apps Interposing System Libraries or Inserting Libraries at Runtime

- Multiple components of Xcode 7 run with library validation enabled in order to prevent exploitation by malicious software. Any software that interposes system libraries or is injected into process using DYLD_INSERT_LIBRARIES may cause problems when running Xcode 7.

  Workaround: You are advised to either remove software that interposes system libraries or is injected into process using DYLD_INSERT_LIBRARIES from your system or contact the vendor of such software for an update. The crash log and syslog will both reference the file that is causing the problem. (22366994)

Debugging

• The "Debug > Simulate Background Fetch" menu item will initiate two background fetch events each time it is selected when debugging an app running on an iOS 9 device. This issue does not affect the simulator or other iOS versions. (22077789)

- Xcode may hang when doing a Quick Look of a variable. (22334782)

watchOS

- The Apple Watch simulator may stop taking input after a reset or reboot.

  Workaround: If the simulator doesn't respond to the Home button, quit and restart the Apple Watch simulator application. (21135676)

- Address sanitizer may not work for watchOS apps. (19789677)

- UI elements on a Watch extension may appear in English despite the app being localized to a different language.

  Workaround: Localize all components of your Watch app, including its extension, to the desired language. (22065581)

- You are not be able to debug a watchOS 1 app extension in a project that also has watchOS 2 app built in the same iOS app.

  Workaround: The system prefers the watchOS 2 app when both are present, so you need to remove it from the iOS app bundle. Edit the Build Phases of the iOS App to remove the watchOS 2 app as a build dependency of the iOS App and remove it from the Embed Watch Content build phase. Clean the build products, and then Run to debug the watchOS 1 app extension. (21173814)

New in Xcode 7 beta 6 - Swift 2.0 and Objective-C

Swift Language Enhancements and Changes

- A new keyword 'try?' has been added to Swift. 'try?' attempts to perform an operation that may throw. If the operation succeeds, the result is wrapped in an optional; if it fails (I.e. if an error is thrown), the result is 'nil' and the error is discarded. 'try?' is particularly useful when used in conjunction with "if let" and "guard".

```
func produceGizmoUsingTechnology() throws -> Gizmo { … }
func produceGizmoUsingMagic() throws -> Gizmo { … }

if let result = try? produceGizmoUsingTechnology() { return result }
if let result = try? produceGizmoUsingMagic() { return result }
print("warning: failed to produce a Gizmo in any way")
return nil
```

It is worth noting that 'try?' always adds an extra level of Optional to the result type of the expression being evaluated. If a throwing function's normal return type is 'Int?', the result of calling it with 'try?' will be 'Int??', or 'Optional<Optional<Int>>'. (21692467)

- Xcode now provides context-sensitive code completions for enum elements and option sets when using the shorter dot syntax. (16659653)

- Static computed properties defined in protocol extensions are now supported. (21358641)

- Variadic parameters can now appear anywhere in the parameter list for a function or initializer. For example:

```
func doSomethingToValues(values: Int..., options: MyOptions = [], fn: (Int) -> Void) { … }
```

(20127197)

- Collections containing types that are not Objective-C compatible are no longer considered Objective-C compatible types themselves. For example, previously `Array<SwiftClassType>` was permitted as the type of a property marked `@objc`. This is no longer the case. (19787270)

- C typedefs of block types are now imported as typealiases for Swift closures. The primary result of this is that typedefs for blocks with a parameter of type BOOL are now imported as closures with a parameter of type `Bool` (rather than `ObjCBool` as in the previous beta). This matches the behavior of block parameters to imported Objective-C methods. (22013912)

- Error messages produced by the type checker continue to improve in specificity and helpfulness, and now print "aka" when a type in a diagnostic involves a type alias. (19036351)

Swift Standard Library Enhancements and Changes

- The `print()` and `debugPrint()` functions were improved:

  1. They became variadic, so you can print any number of items with a single call
  2. We added `separator: String = " "`, so you can control how the items are separated

3 `appendNewline: bool = true` was replaced with `terminator: String = "\n"`. Thus, `print(x, appendNewline: false)` becomes `print(x, terminator: "")`.
4 For the variants that take an output stream, we added the argument label `toStream` to the stream argument.

(21788540)

- The method `RangeReplaceableCollectionType.extend()` was renamed to `appendContentsOf()`, and the `splice()` method was renamed to `insertContentsOf()`. (21972324)

- Most standard library APIs that take closures or @autoclosure parameters now use `rethrows`. This allows the closure parameters to methods like `map` and `filter` to throw errors, and allows short-circuiting operators like &&, ||, and ?? to work with expressions that may produce errors. (21345565)

- All CollectionType's are now sliceable. SequenceType now has a notion of SubSequence, which is a type that represents only some of the values but in the same order. For example, Array's SubSequence type is ArraySlice, which is an efficient view on the Array's buffer that avoids copying as long as it uniquely references the Array from which it came.

  The following free Swift functions for splitting/slicing sequences have been removed and replaced by method requirements on the SequenceType protocol with default implementations in protocol extensions. CollectionType has specialized implementations where possible to take advantage of efficient access of its elements.

```
/// Returns the first `maxLength` elements of `self`,
/// or all the elements if `self` has fewer than `maxLength` elements.
prefix(maxLength: Int) -> SubSequence
/// Returns the last `maxLength` elements of `self`,
/// or all the elements if `self` has fewer than `maxLength` elements.
suffix(maxLength: Int) -> SubSequence
/// Returns all but the first `n` elements of `self`.
dropFirst(n: Int) -> SubSequence
/// Returns all but the last `n` elements of `self`.
dropLast(n: Int) -> SubSequence
/// Returns the maximal `SubSequence`s of `self`, in order, that
/// don't contain elements satisfying the predicate `isSeparator`.
split(maxSplits maxSplits: Int, allowEmptySlices: Bool, @noescape isSeparator:
(Generator.Element) -> Bool) -> [SubSequence]
```

  The following convenience extension is provided for split:

```
split(separator: Generator.Element, maxSplit: Int, allowEmptySlices: Bool) ->
[SubSequence]
```

  Also, new protocol requirements and default implementations on CollectionType are now available:

```
/// Returns `self[startIndex..<end]`
prefixUpTo(end: Index) -> SubSequence
/// Returns `self[start..<endIndex]`
suffixFrom(start: Index) -> SubSequence
/// Returns `prefixUpTo(position.successor())`
prefixThrough(position: Index) -> SubSequence
```

(21663830)

- For consistency and better composition of generic code, `ArraySlice` indices are no longer always zero-based but map directly onto the indices of the collection they are slicing and maintain that mapping even after mutations.

  Before:

```
var a = Array(0..<10)
var s = a[5..<10]
s.indices        // 0..<5
s[0] = 111
s                // [111, 6, 7, 8, 9]
s.removeFirst()
s.indices        // 1..<5
```

  After:

```
var a = Array(0..<10)
var s = a[5..<10]
s.indices        // 5..<10
s[5] = 99
s                // [99, 6, 7, 8, 9]
s.removeFirst()
s.indices        // 6..<10
```

  Rather than define variants of collection algorithms that take explicit subrange arguments, such as `a.sortSubrangeInPlace(3..<7)`, the Swift standard library provides slicing, which composes well with algorithms, so one can write `a[3..<7].sortInPlace()`. With most collections, these algorithms compose naturally:

```
extension MyIntCollection {
  func prefixThroughFirstNegativeSubrange() -> SubSequence {
    // Find the first negative element
    let firstNegative = self.indexOf { $0 < 0 } ?? endIndex

    // Slice off non-negative prefix
    let startsWithNegative = self.suffixFrom(firstNegative)

    // Find the first non-negative position in the slice
    let end = startsWithNegative.indexOf { $0 >= 0 } ?? endIndex
    return self[startIndex..<end]
  }
}
```

  The above code works for any collection of `Ints...` unless it is an `Array<Int>`. Unfortunately, when array slice indices are zero-based, we need to change the last two lines of the method to:

```
let end = startsWithNegative.indexOf { $0 >= 0 }
  ?? startsWithNegative.endIndex
return self[startIndex..<end + firstNegative]
```

  These differences made working with slices awkward, error-prone, and non-generic.

After this change, Swift collections start to provide a guarantee that, at least until there is a mutation, slice indices are valid in the collection from which they were sliced, and refer to the same elements. (21866825)

Resolved in Xcode 7 beta 6 - Swift 2.0 and Objective-C

Swift Debugging

- Swift expressions in LLDB that can throw errors required a `try` prefix in a prior beta. Relaxed syntax for expressions now works as expected. (21990339)
- Evaluating a Swift expression with `po` when debugging arm64 code on devices no longer fails. (22143534)

Swift Language & Compiler

- Use of `guard let` at the top level of a playground or script file no longer results in memory corruption. (21792430)
- The compiler now generates the correct floating-point value when a negative integer literal is used in a generic <`T: IntegerLiteralConvertible`> context with T as a floating-point type. (20467684)
- Invalid expressions involving prefix & no longer crash the compiler. (18496742)
- Fixed a compiler crash when a protocol extension setter is invoked on a class that conforms to the protocol. (21578832)

Known Issues in Xcode 7 beta 6 — Swift 2.0 and Objective-C

Swift Language & Compiler

- Declaring multiple global variables in a single `var` or `let` may cause corruption of their values.(22207407)

  Workaround: Declare each variable using a separate `var` or `let`.

- When extending a Core Foundation type to conform to a protocol, checked casts to the protocol type will fail at runtime in optimized builds. For example:

```
extension CFSet: Fooable {
  func foo() { print("CFSet") }
}

import Foundation

protocol Fooable {
  func foo()
}

func fooify<T>(x: T) {
  if let foo = x as? Fooable {
    foo.foo()
  } else {
    print("not fooable")
  }
}
fooify(CFSetCreate(…))
```

  Workaround: If the Core Foundation type has a toll-free-bridged Objective-C equivalent, extend the Objective-C class to conform to the protocol instead of the Core Foundation type. (20882882)

- Array slices with a non-zero lower index are incorrectly displayed in the Swift REPL and LLDB's "po" output. Playgrounds that produce array slices with non-zero lower indicies will fail with EXC_BAD_INSTRUCTION. (22373053)

Xcode 7 beta 5

New in Xcode 7 beta 5 — IDE

Playgrounds

- Playgrounds can now be set to run manually or automatically when changes are made, as well as stopped during execution. (18058289)

Crash Logs

- Xcode can now display crash reports in the Crashes Organizer that originated from your iOS App Extension crashes. (19309974)

Instruments

- Instruments now supports disabling the Automatic Termination feature of Transparent Application Lifecycle when launching OS X applications.

  See the following link for details on Automatic Termination for OS X apps: https://developer.apple.com/library/mac/documentation/General/Conceptual/MOSXAppProgrammingGuide/CoreAppDesign/CoreAppDesign.html#//apple_ref/doc/uid/TP40010543-CH3-SW27

  To disable Automatic Termination when Instruments launches an OS X app, select the “Edit {app name}…” menu item in the process chooser after initially selecting the process. Select the gear icon in the application settings window, then select the “Disabled” menu item underneath the “Transparent Application Lifecycle - Automatic Termination” section. From there forward, Instruments will launch the OS X app with an additional command line option that disables Automatic Termination. This setting will be remembered for all future launches of that app. (18105188)

Resolved in Xcode 7 beta 5 — IDE

General

- The Info.plist will be regenerated when you change the value of the PRODUCT_BUNDLE_IDENTIFIER build setting and then rebuild. (21066965)

- Known issues have been fixed which caused builds to fail after installing a new version of Xcode. Previously, users had to delete the Derived Data folder to work around these issues. If you continue to experience these sorts of issues, please file bug reports at https://bugreport.apple.com. (22067768)

- Xcode 7 beta 5 resolves issues with building iOS projects and running the iOS simulator on OS X El Capitan beta 6. (22089926)

Playgrounds

- Executing an iOS Playground for the first time should not take substantially longer than subsequent executions. (21163503)

- Inline result controls are now accessible from the Editor menu or by command-clicking the result to display a contextual menu. (21690082)

Testing

- Code Coverage now supports files in static libraries. The source files show up under each binary that links the static library in the Coverage tab of the Test Report. The source editor shows coverage numbers aggregated across all binaries. (21984681)

- Fixed regression in Xcode 7 betas where setting baselines for performance tests did not work. (21840778)

- Xcode's UI Test Recording no longer generates code which fails to build due to variable name collisions. (21153964)

- Xcode's UI Test Recording no longer generates code which fails to build due to calling XCUIElement API on a XCUIElementQuery returned from -containingType:identifier: (21676455)

Localization

- You can import XLIFF files with xcodebuild using the importLocalization command. However, imports with xcodebuild will only update the contents of existing strings files, and cannot add new strings files to projects. When an import needs to add new strings files, xcodebuild will prompt you to import the localizations with Xcode. (20125571)

watchOS

- When building and running your iOS app containing a WatchKit 2.0 app, the installation of the WatchKit 2.0 app will be faster after the initial installation when not much has changed in the app. (21126483)

- Installation of Apple Watch apps is more reliable when trying to debug from Xcode. (21171436)
- The Debug->Attach To Process menu in Xcode lists processes on the Watch. (20962546)

App Thinning

- Xcode will no longer warn that "Application Slicing is disabled for this seed of Xcode if size classes are used." (21278829)

Address Sanitizer

- The address sanitizer now works with app extensions on iOS devices (21118074).

Debugger

- Opening the view debugger no longer crashes Xcode. (21844453)

New in Xcode 7 beta 5 — Swift 2.0 and Objective-C

Linker

- Weak frameworks and weak libraries are now supported when bitcode is enabled. The linker no longer warns about `–weak_framework`, `–weak-l` or `–weak_library` not being able to be used together with `–bitcode_bundle`. (22037565)

Swift Language Enhancements and Changes

- Structs and classes are now allowed to conform to `ErrorType`. (21867608)

- When delegating or chaining to a failable initializer (e.g., with `self.init(…)` or `super.init(…)`), one can now force-unwrap the result with `!`. For example:

```
extension UIImage {
  enum AssetIdentifier: String {
    case Isabella
    case William
    case Olivia
  }

  convenience init(assetIdentifier: AssetIdentifier) {
    self.init(named: assetIdentifier.rawValue)!
  }
}
```

This allows defining non-optional initializers that chain to failable ones. (18497407)

- A new feature has been introduced to improve the performance of -Onone (debug) builds, by using pre-specialized instances of generics in the standard library. It produces significantly faster executables in debug builds in many cases, without impacting compile time. (20486658)

- Generic subclasses of Objective-C classes, as well as non-generic classes that inherit from such a class, require runtime metadata instantiation and cannot be directly named from Objective-C code. Beta 4 added support for defining generic subclasses of Objective-C classes, but the generated Objective-C bridging header would erroneously list such classes. This could lead to incorrect runtime behavior or compile-time errors and is now fixed.

  Furthermore, the behavior of the `@objc` attribute on a class has been clarified. Applying `@objc` to a class which cannot appear in a bridging header is now an error. Note that this will not result in a change of behavior with valid code, because members of a class are implicitly `@objc` if any superclass of the class is an `@objc` class, and all `@objc` classes must inherit from `NSObject`. (21342574)

- The type `Boolean` in `MacTypes.h` is now imported as `Bool` in contexts that allow bridging between Swift and Objective-C types. In cases where the representation is significant (e.g. when you use the C type “Boolean*”), `Boolean` is imported as a distinct `DarwinBoolean` type, which is `BooleanLiteralConvertible` and can be used in conditions (much like the `ObjCBool` type). (19013551)

- The NSManaged attribute can now be used with methods as well as properties, for access to Core Data's automatically-generated Key-Value-Coding-compliant to-many accessors.

```
@NSManaged var employees: NSSet

@NSManaged func addEmployeesObject(employee: Employee)
@NSManaged func removeEmployeesObject(employee: Employee)
@NSManaged func addEmployees(employees: NSSet)
@NSManaged func removeEmployees(employees: NSSet)
```

These can be declared in your NSManagedObject subclass. (17583057)

- Type names and enum cases now print and convert to String without qualification by default. debugPrint or String(reflecting:) can still be used to get fully-qualified names (21788604). For example:

```
enum Fruit { case Apple, Banana, Strawberry }
print(Fruit.Apple) // "Apple"
debugPrint(Fruit.Apple) // "MyApp.Fruit.Apple")
```

- print() and reflection via Mirrors is now able to report both the current case and payload for all enums with multiple payload types. The only remaining enum types that do not support reflection are @objc enums and enums imported from C. (21739870)

Swift Standard Library Enhancements and Changes

- forEach has been added to SequenceType. This lets you iterate over elements of a sequence, calling a body closure on each.

```
(0..<10).forEach {
     print($0)
}
```

This is very similar to the following:

```
for x in 0..<10 {
    print(x)
}
```

but take note of the following differences:

- Unlike for-in loops, you can't use break or continue to exit the current call of the body closure or skip subsequent calls.
- Also unlike for-in loops, using return in the body closure will only exit from the current call to the closure, not any outer scope, and won't skip subsequent calls.
- Due to these limitations, the forEach member is only recommended when applied to a chained series of functional algorithms (e.g. foo.map {…}.filter {… }.forEach { …}) and when the body is small. In other cases, we recommend using the for..in statement. (18231840)

- SIMD: Integer vector types in the simd module now only support unchecked arithmetic with wraparound semantics using the &+, &−, and &∗ operators, in order to better support the machine model for vectors. The `+`, `-`, and `*` operators are unavailable on integer vectors and Xcode will automatically suggest replacing them with the wrapping operators.

- SIMD: Code generation for vector types in the simd module has been improved to better utilize vector hardware, leading to dramatically improved performance in many cases.
(21574425)

- Dictionary's `removeAtIndex(_:)` method now returns the key-value pair being removed as a two-element tuple (rather than returning Void). Similarly, Set's `removeAtIndex(_:)` returns the element being removed. (20299881)

- The `Word` and `UWord` types are removed from the standard library, use Int and UInt instead. (18693488)

Resolved in Xcode 7 beta 5 — Swift 2.0 and Objective-C

Swift Migrator

- “Convert to Latest Swift” no longer generates spurious build errors when run. (19650497)

Swift Documentation

- Extended documentation for the Swift standard library is now included in the QuickHelp popover. (21234431)

Swift Language & Compiler

- Incremental builds: changing just the body of a function should no longer cause dependent files to be rebuilt. (15352929)
- Type checker diagnostics continue to improve in precision and specificity. (19870975)
- `if case` and `guard case` patterns no longer crash when applied to indirect enums. (21948603)
- The compiler now generates the correct floating-point value when a negative integer literal is used in a generic <`T: IntegerLiteralConvertible`> context with T as a floating-point type. (21948603)
- In OS X Cocoa apps, subclasses of `NSView` have a `print` method which shadows the global Swift `print` function. Referring to `print` inside an `NSView` subclass now tells you to use the `Swift.print` global function or refer to “self.print” explicitly. (18309853)
- The compiler no longer crashes when a generic struct contains weak or unowned references. (21315113)
- A bug has been fixed that caused an LLVM error “Terminator found in the middle of a basic block” to be raised when generating code for some `switch` statements over enums with a single associated value. (21742335)
- When Objective-C protocol type names are printed, they no longer lose the first three letters of their names. (21887583)

Xcode 7 beta 4

New in Xcode 7 beta 4 — IDE

Interface Builder

- Storyboard references may be deployed to iOS 8, OS X 10.10, and watchOS 1. Please note that backwards-deployed storyboard references may not be connected to relationship segues and may not refer to storyboards in external bundles. (21275172)

Testing

- Using UI recording generates code for iOS gestures like `doubleTap`, `twoFingerTap`, `longPress`, `swipeUp`, `swipeDown`, `swipeLeft`, and `swipeRight`. (20278248)

Resolved in Xcode 7 beta 4 — IDE

General

- Xcode supports deploying apps on devices with iOS 8.4. (21640571, 20699475, 21107693)

Code Coverage

- Code coverage works with UI testing. (20966994)

Testing

- The OCUnit to XCTest migrator does not leave test targets linking against SenTestingKit. (21235034)
- Long assertions in the test report are not truncated. (21083089)
- Unpredictable results caused by Auto-correct in UI tests that type text have been fixed. (21106884)
- UI testing finds elements with names containing decomposable Unicode characters. (20804391)

Debugger

- Breakpoints set in source code for the target application that had not worked in previous betas of Xcode 7 should work as expected starting with Xcode 7 beta 4. (21798743)
- GPU debugger frame captures of iOS Metal apps no longer crash Xcode. (21741213)

Source Control

- Stability issues with Blame and Log SCM editor modes have been resolved. (21487571)

Interface Builder

- Storyboard references work with WatchKit storyboards. (21191010)
- The Interface Builder connections inspector properly shows a view outlet for Swift ViewController subclasses if they are marked with `@objc` but do not have an exported class name. (20909753)

Playgrounds

- The erratic cursor behavior seen in playground documents stored in folders managed by network sync-service has been fixed. (21358571)

New in Xcode 7 beta 4 — Swift 2.0 and Objective-C

Objective-C Language Changes

- The double-underscored nullability qualifiers (`__nullable`, `__nonnull`, and `__null_unspecified`) have been renamed to use a single underscore with a capital letter (`_Nullable`, `_Nonnull`, and `_Null_unspecified`, respectively). The compiler predefines macros mapping from the old double-underscored names to the new names for source compatibility. (21530726)

Swift Language Changes

- Enums and cases can be marked `indirect`, which causes the associated value for the enum to be stored indirectly, allowing for recursive data structures to be defined. For example:

```
enum List<T> {
  case Nil
  indirect case Cons(head: T, tail: List<T>)
}

indirect enum Tree<T> {
  case Leaf(T)
  case Branch(left: Tree<T>, right: Tree<T>)
}
```

  (21643855)

- AnyObject and NSObject variables that refer to class objects can be cast back to class object types. For example, this now succeeds:

```
let x: AnyObject = NSObject.self
let y = x as! NSObject.Type
```

- Arrays, dictionaries, and sets that contain class objects successfully bridge with NSArray, NSDictionary, and NSSet as well. Objective-C APIs that provide NSArray<Class> * objects, such as -[NSURLSessionConfiguration protocolClasses], now work correctly when used in Swift. (16238475)

- Applying the `@objc` attribute to a class changes that class's compile-time name in the target's generated Objective-C header, as well as changing its runtime name. This applies to protocols as well. For example:

```
// Swift
@objc(MyAppDelegate)
class AppDelegate : NSObject, UIApplicationDelegate {
  // ...
}
// Objective-C
@interface MyAppDelegate : NSObject <UIApplicationDelegate>
  // ...
@end
```

  (17469485)

- The ability to refer to the "0" element of a scalar value (producing the scalar value itself) has been removed. (17963034)

- The error message diagnostics produced by the type checker constraint system are somewhat improved in this beta. Further improvements are coming in a subsequent beta. (20806331)

- The `SinkType` protocol and `SinkOf` struct have been removed from the standard library in favor of `(T) -> ()` closures. (21663799)

- `ExtensibleCollectionType` has been folded into `RangeReplaceableCollectionType`. In addition, default implementations have been added as methods, which should be used instead of the free Swift module functions related to these protocols. (18220295)

- Properties and methods using `Unmanaged` can be exposed to Objective-C. (16832080)

- The `performSelector` family of APIs is now available for Swift code. (17227475)

- Fields of C structs that are marked `__unsafe_unretained` are presented in Swift using Unmanaged. It is not possible for the Swift compiler to know if these references are really intended to be strong (+1) or unretained (+0). (19790608)

- Swift documentation comments recognize a top-level list item - `Throws: ...` which should be used to document what errors can be thrown and why. These appear alongside parameters and return descriptions in Xcode's QuickHelp. (21621679)

- Types can conform to protocols that are less available than the type itself. For example:

```
@available(iOS 9.0, *)
  protocol P { … }

@available(iOS 7.0, *)
class MyController : UIViewController, P {
  ...
}
```

(21718497)

Resolved Issues in Xcode 7 beta 4 — Swift 2.0 and Objective-C

Swift Language & Compiler

- When throwing a reference to an `NSError` instance in Swift, the Swift runtime no longer loses the userInfo of the original `NSError` if it is caught as an `NSError`. The Swift runtime now preserves the identity of the original `NSError`. For example, this assertion now holds:

```
let e = NSError(…)
do {
  throw e
} catch let e2 as NSError {
  assert(e === e2)
}
```

  (21546914)

- The Swift runtime metadata cache crash when trying to form very large tuple types has been fixed. (21659505)

- The Swift runtime crash when a protocol conformance is added to an unavailable weakly-linked class via an extension has been fixed. (21541766)

- Xcode handles auto-indentation correctly inside inactive blocks of #if config statements. (16427856)

Xcode 7 beta 3

New in Xcode 7 beta 3 — IDE

Build System

- The Xcode build system no longer automatically inherits the environment used to launch the app when running in the IDE. This prevents unnecessary rebuilds when Xcode.app is opened from the command line. If necessary, users can opt-in to the old behavior by entering this command in Terminal:

```
defaults write com.apple.dt.Xcode UseSanitizedBuildSystemEnvironment -bool
NO
```

  NOTE: For compatibility reasons, this behavior is not the default when running builds via xcodebuild. However, users who wish to interchange between building Xcode and building using xcodebuild can add the following argument:

```
-UseSanitizedBuildSystemEnvironment=YES
```

  on the command line to also use a sanitized environment in xcodebuild. (20668232)

- Xcode now allows the IDEBuildOperationMaxNumberOfConcurrentCompileTasks user default to exceed the number of CPUs available in the machine. (20998547)

- Module debugging in clang can now be disabled in individual targets via the build setting "Enable Clang Module Debugging". This should only be disabled when building static libraries with debug info for distribution. (21385543)

- Xcode now populates build settings during the build which describe the version of the SDK being build against. These settings are:
  - SDK_VERSION: x.y version of the SDK being used. For example, '10.10' for OS X 10.10, or '8.0' for iOS 8.
  - SDK_VERSION_ACTUAL: Single integer representing the SDK version. For example, 101000, or 80000.
  - SDK_VERSION_MAJOR: Single integer representing the major version of the SDK. For example, 101000 (which would be the value for 10.10.0 or 10.10.4), or 80000 (which would be the value for 8.0 or 8.3).
  - SDK_VERSION_MINOR: Single integer representing the minor version of the SDK. For example, 1000 or 1004 (for 10.10.0 or 10.10.4), or 0 or 300 (for 8.0 or 8.3).

  (20341285)

- Stale file removal has been modified to only apply to the “build” action when using xcodebuild. (21568261)

App Store

- Exporting an archive for the App Store now requires an account with iTunes Connect Access. (19940553)

Testing

- The XCUIElementQuery API, elementAtIndex, has been deprecated and replaced with elementBoundByIndex. Calls to elementAtIndex can be replaced with calls to elementBoundByIndex. (21566073)

Resolved Issues in Xcode 7 beta 3 — IDE

Metal

- Shader profiling is available when capturing a frame from Metal applications using Nvidia GPUs. (21165399)
- texturetool now uses the same vertical orientation for ASTC compressed textures as used for other compressed texture formats. Textures previously compressed to ASTC with the iOS 8 SDK & recompressed to ASTC with the iOS 9 SDK will need to have their vertical orientation flipped. (20755603)

Build System

- Deployment of 64-bit-only iOS apps is supported to devices running iOS versions earlier than 8.0. (21195657)
- The Xcode build system no longer limits the amount of output from shell script build phases and shell script build rules to only 200 messages. (16471986)
- xcodebuild no longer reports an error when attempt to build using two different action kinds (e.g., “build” versus “install”) in the same directories. (21568261)

Interface Builder

- Distance guides, which are displayed when holding the Option key while hovering over a view hierarchy in Interface Builder, correctly appear when using OS X 10.11. (20758551)
- The Interface Builder Connections Inspector properly shows a view outlet for Swift ViewController subclasses if they are marked with @objc but do not have an exported class name. (20909753)

Testing

- The OCUnit to XCTest migrator no longer leaves test targets still linking against SenTestingKit. (21235034)
- UI recording’s generated code for Swift is correct for multiple modifiers. (21183420)

Debugging

- When debugging your view hierarchy, the inspector will update when selecting views in the debug navigator. (20748069)
- You can now inspect results and edit target criteria after running performance tests within Xcode. (21252208)

Simulator

- Printing from Simulator works with the iOS 9.0 beta runtime. (19754468)

- Xcode 7.0 beta now supports downloadable legacy simulators. (20699475)

watchOS

- When performing the Profile action from Xcode on a WatchKit App the first time, Instruments launch the WatchKit App properly. (21241660)
- Storyboard References now work with WatchKit storyboards. (21126778)
- When using Xcode or Instruments, there may be times when an Apple Watch device is listed as offline even though the companion iPhone is attached to the computer. (21104615)
- If your home directory is on a case-sensitive filesystem, native watch app debugging is now supported. (21123503)
- Building and running as glance will now work even if glance isn't enabled in the watchOS 2.0 app. (21188494)

New in Xcode 7 beta 3 — Swift 2.0 and Objective-C

Swift Language Changes

- Formatting for Swift expression results has changed significantly in this beta when using "po" or "expr -O". Customization that was introduced in the first beta has been refined in the following ways:

  - The formatted summary provided by either debugDescription or description methods will always be used for types that conform to CustomDebugStringConvertible or CustomStringConvertible respectively. When neither conformance is present the type name will be displayed, and reference types will also display the referenced address to more closely mimic existing behavior for Objective-C classes.
  - Value types such as enums, tuples, and structs will display all members indented below the summary by default, while reference types will not. This behavior can be customized for all types by implementing CustomReflectable.

  These output customizations can be bypassed by using "p" or "expr" without the -O argument to provide a complete list of all fields and their values. (21463866)

If an element of an enum with string raw type does not have an explicit raw value, it will default to the text of the enum's name. For example,

```
enum WorldLayer : String {
    case Ground, BelowCharacter, Character
}
```

is equivalent to

```
enum WorldLayer : String {
    case Ground = "Ground"
    case BelowCharacter = "BelowCharacter"
    case Character = "Character"
}
```

(15819953)

- Unnamed parameters now require an explicit `_:` to indicate that they are unnamed. For example, the following is now an error:

  ```
  func f(Int) { }
  ```

  and must be written as

  ```
  func f(_: Int) { }
  ```

  This simplifies the argument label model, and also clarifies why cases like `func f((a: Int, b: Int))` do not have parameters named `a` and `b`. (16737312)

- It is now possible to append a tuple to an array. (17875634)

- Generic subclasses of Objective-C classes are now supported. (18505295)

- The grammar has been adjusted so that lines beginning with `.` are always parsed as method or property lookups following the previous line, allowing for code formatted like this to work:

```
foo
  .bar
  .bas = 68000
```

  It is no longer possible to begin a line with a contextual static member lookup (for example, to say `.staticVar = MyType()`). (20238557)

- Errors trapped by a `try!` expression, or unhandled at the top level of a script or playground, now display the error value in the trap message. (20807523)

- The `NS_REFINED_FOR_SWIFT` macro can be used to move an Objective-C declaration aside to provide a better version of the same API in Swift, while still having the original implementation available. (For example, an Objective-C API that takes a Class could offer a more precise parameter type in Swift.) (20070465)

  The `NS_REFINED_FOR_SWIFT` macro operates differently on different declarations. For example:

  - Init methods will be imported with the resulting Swift initializer having `__` prepended to its first external parameter name.

```
- (instancetype)initWithClassName:(NSString *)name NS_REFINED_FOR_SWIFT;

init(__className: String)
```

  - Other methods will be imported with `__` prepended to their base name.

```
- (NSString *)displayNameForMode:(DisplayMode)mode NS_REFINED_FOR_SWIFT;

func __displayNameForMode(mode: DisplayMode) -> String
```

  - Subscript methods will be treated like any other methods and will not be imported as subscripts.

  - Other declarations will have `__` prepended to their name.

```
@property DisplayMode mode NS_REFINED_FOR_SWIFT;

var __mode: DisplayMode { get set }
```

- Generic parameters on types in the Swift standard library have been renamed to reflect the role of the types in the API. For example, `Array<T>` became `Array<Element>`, `UnsafePointer<T>` became `UnsafePointer<Memory>` etc. (21429126)

Resolved Issues in Xcode 7 beta 3 — Swift 2.0 and Objective-C

Swift Migrator

- If the migrator is run on code that already migrated to 2.0 and it fails to detect Swift 2.0 syntax, a warning will now be shown. (20882092)

Swift Language & Compiler

- The `guard` statement now binds variable names properly at the top level of a playground, script file, or in `main.swift`. (21110580)

- A crash when a subclass overrides a method that takes a non-optional closure parameter with one that takes an optional closure parameter has been fixed. (21154055)

  For example, this no longer crashes:

```
class Foo {
  func foo(x: Int -> Int) {}
}
class Bar: Foo {
  override func foo(x: (Int -> Int)?) {}
}
```

- The optimizer no longer removes `try!` traps when runtime checks are not disabled. (21414222)

- Protocol extensions that place an additional protocol requirement on `Self`, e.g.,

```
extension Drawable where Self : Equatable {
   // ...
  }
```

  now provide appropriate overload resolution and several related compiler crashes have been fixed. (21401180)

- The Objective-C metadata generated for `@objc` properties defined in Swift now correctly refers to the fully-qualified names of Swift classes and protocols, making the results suitable for use with `NSClassFromString` et al. (21368142)

  For example:

```
class Foo : NSObject {
   var prop: Foo? // Objective-C property now refers to "<modulename>.Foo"
rather than "Foo"
  }
```

Xcode 7 beta 2

New in Xcode 7 beta 2 — Swift 2.0 and Objective-C

Swift Language Changes

- Enum cases with payloads can now be used as functions. (19091028)

  For example:

```
enum Either<T, U> { case Left(T), Right(U) }
let lefts: [Either<Int, String>] = [1, 2, 3].map(Either.Left)
let rights: [Either<Int, String>] = [“one”, “two”,
    “three”].map(Either.Right)
```

- Non-mutating methods of structs, enums, and protocols may now be partially applied to their “self” parameter. (21091944)

  For example:

```
let a: Set<Int> = [1, 2, 3]
let b: [Set<Int>] = [[1], [4]]
b.map(a.union) // => [[1, 2, 3], [1, 2, 3, 4]]
```

- Initializers can now be referenced like static methods by referring to `.init` on a static type reference or type object:

```
let x = String.init(5)
let stringType = String.self
let y = stringType.init(5)

let oneTwoThree = [1, 2, 3].map(String.init).reduce(“”, combine: +)
```

  `.init` is still implicit when constructing using a static type, as in `String(5).` `.init` is required when using dynamic type objects, or when referring to the initializer as a function value. (21375845)

- Code generation for large struct and enum types has been improved to reduce code size. (20598289)

- Swift is now built with assertions enabled. This will help generate actionable crash reports.

Xcode 7 beta 1

New Features in Xcode 7 beta - IDE

Swift in Xcode

- Playgrounds support multiple pages. You can add a new page to any playground by selecting File > New Playground Page. In the navigator and jump bar, you'll see all pages in the playground. You can add navigation between pages with the new page navigation markup (20192277):

  Navigation relative to pages in the playground:
  [Go to First Page](@first)
  [Go to Last Page](@last)

  Navigation relative to the current page:
  [Go to Next Page](@next)
  [Go to Previous Page](@previous)

  Navigation to a specific page:
  [Go to Overview](Overview)
  [Go to Sorting](Sorting)

- You can use the assistant editor to get a summary of the interface of your Swift classes. When viewing a Swift file in the primary editor, the Counterparts assistant will show a summarized version of the class that includes declarations (but not implementation) of the methods and functions in your file along with documentation comments. (17684981)

- A migrator is available to help migrate your code from Swift 1.2 to Swift 2.0. Select Edit->Convert->To Latest Swift Syntax. The migrator is fully effective when it is applied on Swift 1.2 projects. If the migrator detects that the code is fully or partially migrated to Swift 2.0, it will only perform simple changes to correct certain compiler diagnostics. (19653306)

- "Generated Interface" of the assistant editor can be used to get the Swift interface of an Objective-C header file in your own project. (19320817)

- Xcode 7 has a optimization-level menu for Swift under ‘build settings’. It is possible to select Whole-Module optimization from the optimization-level menu. Previously, unchecked build were an optimization level.

- Use the “disable safety checks” build setting to disable runtime checks.

- Compiling Swift projects in Whole Module Optimizations mode will parallelize some compilation phases, which reduces compile time.

Interface Builder

- Default keyboard shortcuts for zooming in Interface Builder have been changed to match the rest of Xcode. (15057238, 16530809)

- Interface Builder adds the ability to edit auxiliary objects belonging to a storyboard scene. (9478187)

- Interface Builder supports the new NSCollectionView API in 10.11, using dataSource and layouts like on iOS. (18997303)

    Workaround: It is not possible to configure prototype items in Interface Builder. Use -[NSCollectionView registerClass:forItemWithIdentifier:] or -[NSCollectionView registerNib:forItemWithIdentifier:] in code.

- Notes from Interface Builder's Identity inspector are included in --export-strings-file output and XLIFF files exported using the project editor's Export For Localization feature. (18023555)

- Interface Builder supports placeholder references for scenes in other storyboards, and segues that cross storyboard boundaries. (9565583)

- Interface Builder can set multiple left/right bar button items in iOS projects. (10293104)

- Interface Builder enables setting the accessiblityIdentifier property of AppKit and UIKit views, via the Identity inspector. (8913778, 20377226)

- Interface Builder adds authoring support for the new UIStackView (iOS 9) and exposes a distribution property for NSStackView (OS X 10.11). Access this with the "Embed In Stack View" button at the bottom right of the canvas. (18420765)

- Background placeholders on custom views and other containers can be hidden on the Interface Builder canvas with the option "Editor->Canvas->Show Background Placeholders".

- Resize knobs for views on the canvas are interactable outside the canvas frame (15864964)

- Background placeholders on custom views and other containers can be hidden on the Interface Builder canvas with the option "Editor->Canvas->Show Background Placeholders". (20580948)

- Interface builder adds additional constraint types, including proportional width and height constraints in superviews (such as the content view of a window, or view of a view controller). (17011782)

- Interface Builder documents with a development target <= Xcode 4.6 are now upgraded when opening. (12984639)

- Alignment and distribution commands in Interface Builder can be used without generating constraints. (19223827)

- Interface Builder allows subclassing all segue types. (199115040)

- Interface Builder can create multiple storyboards for WatchKit interfaces. (19781408)

Compiler

- The dsymutil utility will fully resolve the type references and .dSYM bundles are entirely self-contained. (14051536)

- The dsymutil utility can unique redundant C++ debug types based on C++'s one-definition-rule. This makes significant size reductions in .dSYM bundles for large C++ projects while preserving the same information. (11659111)

- Clang modules and precompiled headers for C, C++, Objective-C, and Objective-C++ contain debug information for the types they define. When building with the Xcode setting CLANG_ENABLE_MODULE_DEBUGGING=YES (enabled by default), clang stores references to the types defined in the modules instead of their full definitions. This makes significant size reductions in build artifacts and .dSYM bundles.

Build System

- The Xcode build system supports stale file removal of some types of build artifacts which were produced in a previous build, but have since been removed from the project. (19479602)
    - Stale files are removed when the next build detects that they are no longer needed.
    - Stale files will appear in the build log in the reports navigator
    - If removal of a stale file would cause its containing directory to become empty, the directory will also be removed.
    - The kinds of build artifacts which are currently handled by stale file removal are: copied resources and header files, on-demand resources in asset packs, module map specifications, and Swift generated API headers.
    - Only build artifacts produced in the OBJROOT, SYMROOT, or DSTROOT are available for stale file removal.

- The new Product Bundle Identifier build setting (PRODUCT_BUNDLE_IDENTIFIER) is the recommended place to set the Bundle Identifier for a target. The target's Info.plist should be configured to use this build setting by referencing it as "$(PRODUCT_BUNDLE_IDENTIFIER)" in the value for the CFBundleIdentifier key. Xcode will offer to configure this for you when you accept the "Upgrade to recommended settings" project modernization in the issue navigator, unless your target preprocesses its Info.plist, in which case you will need to configure this manually. This change is backwards-compatible to older versions of Xcode. This change is required to make certain features work, such as On Demand Resources, if your target preprocesses its Info.plist. (20887827)

- It is possible to specify per-file compiler flags for derived source files in the "Build Rules" tab of the target editor. These flags will work in Xcode 6.3 as well, though they cannot be edited or viewed with any Xcode prior to Xcode 7. (5924168)

- It is possible to make Xcode run the "unifdef" tool on your headers during a "Copy Headers" build phase. This can be enabled with the COPY_HEADERS_RUN_UNIFDEF build setting, and the flags to pass are controlled by the COPY_HEADERS_UNIFDEF_FLAGS build settings. See "man unifdef" for more information on what flags unifdef accepts. (18786269)

- The user defaults domain for xcodebuild has changed to "com.apple.dt.xcodebuild". xcodebuild will continue to read from the old user defaults domain ("xcodebuild"), albeit at a lower priority than the new one. (20181486)

- "xcodebuild install" command used with “-scheme" builds the targets specified for the "Archive" action of that scheme rather than the "Run" action. (19391445)

- Archive build operations use the “install” value for the “ACTION” build setting. (20192652)
    - This can be used by shell scripts to detect when Xcode is archiving versus building.
    - Existing shell scripts which inspect the ACTION environment variable should be updated according.
    - External build system targets will be passed the “install” action instead of “build”, when archiving, and should be updated accordingly.

    - External build system targets and aggregate targets with shell script build phases will have the appropriate values of TARGET_BUILD_DIR and BUILT_PRODUCTS_DIR which point to appropriate installation location (and respect the SKIP_INSTALL build setting).

- xcodebuild supports a "-hideShellScriptEnvironment" option which causes Xcode to suppress listing the environment variables set for each Run Script build phase. This reduces build output verbosity for some projects. (20309279)

- The "Objective-C Generated Interface Header Name" (SWIFT_OBJC_INTERFACE_HEADER_NAME) build setting can be used to control the name used for the header that is generated by the Swift compiler for use in #import statements in Objective-C. (18063617)

App Thinning

Xcode introduces support for App Thinning: three complementary technologies which deliver an optimized installation automatically.

Bitcode**.** Archive for submission to the App Store in an intermediate representation, which is compiled to the executable for installation on each device. (18168642)

- Xcode 7 has a `ENABLE_BITCODE` option to embed bitcode in apps, app extensions, and frameworks. The option is turned on by default for iOS and is mandatory for watchOS projects submitted to the store.

  When bitcode is enabled for a target, all the objects, static libraries and user frameworks used when linking that target must contain bitcode. Otherwise, an error or a warning will be issued by the linker. (Note: missing bitcode is currently a warning for iOS, but it will become an error in an upcoming beta release of Xcode 7.) `ENABLE_BITCODE` should be consistently turned on for all the targets. If you use a library or framework provided by a third party, please contact the vendor for an updated version which contains bitcode.

  Some compiler and linker options are not supported when bitcode is enabled (e.g., `-mglobal-merge` for clang and `-sectalign` for linker). Some linker options will result in warnings (e.g., `-weak_framework` and `-weak_libraries`). Bitcode builds for iOS also require a minimum deployment target of 6.0. (18447465)

Slicing**.** The App Store tailors the contents of the Asset Catalog for installation on each device.

- On Demand Resources. Host additional content for you app on the iTunes App Store, allowing it to fetch resources as needed using asynchronous download and installation.

Debugger

- Object inspectors for NSView and UIView are available when viewing View hierarchies in the debugger. (18120189, 18124594)

- Address Sanitizer. Objective-C and C/C++ code is particularly susceptible to memory corruption issues such as stack and heap buffer overruns, use-after-free errors, and so forth. They can result in crashes or odd program behavior and could also lead to security exploits. Memory corruption bugs are difficult to track down because they are often hard to reproduce and the root cause can be far from the manifestation of the problem. When you enable the Address sanitizer in your Scheme Editor, Xcode 7 builds your app with added instrumentation so that these issues can be caught immediately, when they happen. Address sanitizer allows

you to inspect the problem right at the place where the memory violation occurs; in many cases, it also provides other information needed to diagnose the bug such as relation of the faulty address to a valid object on the heap and its allocation/deallocation information. Address sanitizer has a small performance overhead; it is fast enough to be used with interactive applications. Address sanitizer is supported on OS X, in the iOS Simulator, and on iOS device. (10514936)

Runtime interposition allows address sanitizer to find bugs in code that is not being recompiled with runtime sanitization enabled. If you run into an issue in external frameworks, we recommend immediately reporting it so that it gets addressed. (Please, add tag "#ASan" in the subject line when filing bugs against Apple frameworks through radar.) However, you can use the following suppression mechanism to unblock yourself and continue on with the testing. This suppression mechanism should only be used for suppressing issues in external code; it does not work on code recompiled with address sanitizer. To suppress errors in external frameworks, set the ASAN_OPTIONS environment variable to point to a suppression file. You can either specify the full path to the file or the path of the file relative to the location of your executable.

```
ASAN_OPTIONS = suppressions=MyASan.supp
```

Use the following format to specify the names of the functions or libraries you want to suppress. You can see these in the error report. Remember that the narrower the scope of the suppression, the more bugs you will be able to catch.

```
interceptor_via_fun:NameOfCFunctionToSuppress
interceptor_via_fun:-[ClassName objCMethodToSuppress:]
interceptor_via_lib:NameOfTheLibraryToSuppress
```

- While debugging your view hierarchy, Xcode allows you to focus on a portion of your app by double clicking on a view. To exit, click the (x) in the debug navigator or double click in the white space of the editor. (18553133)

- Data formatting for Swift types examined using “po” can be customized by implementing one or more new protocols: CustomStringConvertible, CustomDebugStringConvertible, CustomPlaygroundQuickLookable, and CustomReflectable. (18681736)

- Clang modules imported with “expr @import <modulename>” include access to macros and always-inlined functions like NSMakeRect(). (19575787)

- Swift expressions that throw errors in LLDB expression evaluation and Swift REPL sessions produce variables prefixed with $E that can be referenced in subsequent expressions. The expression evaluator also allows a relaxed syntax that does not require try before an expression that can throw. (20864812)

- Vector types in C and Swift are automatically formatted. (20294199)

- Address sanitizer integration is supported through the new “memory” command in LLDB. (21245519)

- Information about inheritance and members of types is available from the LLDB prompt with the “type lookup” command. (17307067)

- The energy gauge is available to iOS apps running on a device. (15032678)

Instruments

- Core Location instrument. This instrument helps users understand CoreLocation usage and tracking the energy impact of outstanding CoreLocation requests by providing backtraces for location-related API use and categorizes active requests by precision.

- Metal System Trace template. A template has been introduced to provide insight into the Metal pipeline and to help understand and optimize the timing of Metal-based applications.

- Track view has been completely overhauled for better performance and more natural user interaction. The track view has an improved thread strategy representation for System Trace, keeps a single selected instrument visible when viewing Thread/CPU strategies to allow for better correlation, and supports more gestures including a smooth pinch-to-zoom.

- Time Profiler is supported for profiling WatchKit Apps on-device. Time Profiler currently only supports deferred mode recordings on Apple Watch to avoid observer effects incurred by profiling wirelessly.

Testing

- UI Testing. Xcode 7 introduces UI testing as a major new feature of the existing XCTest framework. UI testing is implemented as an extension to the existing APIs and concepts in XCTest, making it easy to adopt for developers who have familiarity with Xcode's testing features. (10975541)

- Code Coverage. Visualize the completeness of your test suite by enabling code coverage for your scheme. The code coverage tab in the test report shows which files, functions and lines of code were exercised and, more importantly, which were not exercised. The source code editor can also show code coverage information inline, allowing you to see at a glance which lines—and parts of a line—the tests exercised. (3555436)

Find Navigator

- Xcode lets you find symbols via the "Find > Find Selected Symbol in Project" menu item. (15573153)

- Find navigator shows caller hierarchy. Select a function or method, and right-click to "Find Call Hierarchy" (15716883)

- The Find navigator also displays the language for .strings files containing matches. (18372154)

Schemes

- Users can toggle all the checkboxes in a column in the Manage Schemes sheet by holding down the option key when clicking a checkbox. (8096575)

New in Xcode 7 beta — Swift 2.0 and Objective-C

Swift Language Features

- Error handling. You can create functions that throw, catch, and manage errors in Swift. You can surface and deal with recoverable errors, like “file-not-found” or network timeouts. Swift’s error handling interoperates with `NSError`. (17158652)

- Availability checking. Swift reports an error at compile time if you call an API that was introduced in a version of the operating system newer than the currently selected deployment target. To check whether a potentially unavailable API is available at run time, use the new `#available()` condition in an `if` or `guard` statement. (14586648)

  For example:

```
if #available(iOS 8.0, OSX 10.10, *) {
  // Use Handoff APIs when available.
  let activity =
      NSUserActivity(activityType:"com.example.ShoppingList.view")
  activity.becomeCurrent()
} else {
  // Fall back when Handoff APIs not available.
}
```

- You can specify availability information on your own declarations with the `@available()` attribute. (20938565)

  For example:

```
@available(iOS 8.0, OSX 10.10, *)
func startUserActivity() -> NSUserActivity {
  ...
}
```

  indicates that the `startUserActivity()` method is available on iOS 8.0+, on OSX 10.10+, and on all versions of any other platform.

- Protocol extensions. Extensions can be written for protocol types. This allows methods and properties to be added to any type that conforms to a particular protocol, allowing you to reuse more of your code. This leads to more natural caller side “dot” method syntax that follow the principle of “fluent interfaces” and makes the definition of generic code simpler (reducing “angle bracket blindness”). (11735843)

- Protocol default implementations: Protocols can have default implementations for requirements specified in a protocol extension, allowing “mixin” or “trait” like patterns.

- New `defer` statement. This statement runs cleanup code when the scope is exited, which is particularly useful in conjunction with the new error handling model. (17302850)

  For example:

```
func xyz() throws {
  let f = fopen("x.txt", "r")
  defer { fclose(f) }
  try foo(f)                    // f is closed if an error is propagated.
  let f2 = fopen("y.txt", "r")
```

```
    defer { fclose(f2) }
    try bar(f, f2)                  // f2 is closed, then f is closed if an error is propagated.
}  // f2 is closed, then f is closed on a normal path
```

- New `guard` statement: This new statement allows you to model an early exit out of a scope. For example:

```
guard let z = bar() else { return }
use(z)
```

  The `else` block is required to exit the scope (e.g. with `return`, `throw`, `break`, `continue`, etc) or end in a call to a `@noreturn` function. (20109722)

- Improved pattern matching. `switch/case` pattern matching is available to many new conditional control flow statements, including `if/case`, `while/case`, `guard/case`, and `for-in/case`. `for/in` statements can also have '`where`' clauses, which combine to support many of the features of list comprehensions in other languages.

- New `do` statement: Scopes can be introduced with the `do` statement.

  For example:

```
do {
  //new scope
  do {
    //another scope
  }
}
```

- Testability. Tests of Swift 2.0 frameworks and apps are written without having to make internal routines public. Use `@testable import {ModuleName}` in your test source code to make all public and internal routines usable. The app or framework target needs to be compiled with the "Enable Testability" build setting on. The "Enable Testability" build setting should only be used in your Debug configuration, as it will prohibit optimizations that depend on not exporting internal symbols from the app or framework. (17732115)

- Native support for C function pointers: C functions that take function pointer arguments can be called using closures or global functions, with the restriction that the closure must not capture any of its local context. (16339559)

  For example, the standard C `qsort` function can be invoked as follows:

```
var array = [3, 14, 15, 9, 2, 6, 5]
qsort(&array, array.count, sizeofValue(array[0])) { a, b in
  return Int32(UnsafePointer<Int>(a).memory - UnsafePointer<Int>(b).memory)
}
print(array)
```

- Improved diagnostics: A new warning has been introduced to encourage the use of `let` instead of `var` when appropriate, a warning has also been introduced to about unused variables (15975935), invalid mutation diagnostics are more precise, unreachable switch cases cause a warning, and the switch statement exhaustiveness checker is smarter (20130240).

- SIMD Support:Clang extended vectors are imported and usable in Swift, enabling many graphics and other low-level numeric APIs (e.g. `simd.h`) to be usable in Swift.

- Enums support multiple generic associated values. (15666173)

For example:

```
enum Either<T, U> {
  case Left(T), Right(U)
}
```

- Printing values of certain enum types shows the enum case and payload, if any. This is not supported for @objc enums or certain enums with multiple payloads. (18334936)

- Public extensions of generic types are permitted. (16974298)

  For example:

  ```
  public extension Array { … }
  ```

- Non-generic classes may inherit from generic classes. (15520519)

- Concatenation of Swift string literals, including across multiple lines, is a guaranteed compile-time optimization, even at -Onone. (19125926)

- Failable convenience initializers are allowed to `return nil` before calling `self.init`. (20193929) Designated initializers still must initialize all stored properties before returning nil, which is a known limitation.

- Nested functions can recursively reference themselves and other nested functions. (11266246)

- `if` statements can be labeled, and labeled `break` statements can be used as a jump out of the matching `if` statement. Note that an unlabeled `break` does not exit the `if` statement, it exits the enclosing loop or switch statement, or is illegal if there is none. (19150249)

- A new `x?` pattern can be used to pattern match against optionals as a synonym for `.Some(x)`. (19382878)

- A new `@nonobjc` attribute is introduced to selectively suppress ObjC export for instance members that would otherwise be `@objc`. (16763754)

- A new `readLine()` function has been added to the standard library. (15911365)

Swift Language Changes

- The OS X 10.11, iOS 9, and watchOS 2 SDKs have adopted modern Objective-C features like nullability, typed collections, and others to provide an improved Swift experience.

- The standard library moved many generic global functions (such as `map`, `filter`, and `sort`) to be methods written with protocol extensions. This allows those methods to be pervasively available on all sequence and collection types and allowed the removal of the global functions.

- Methods and functions have the same rules for parameter names. You can omit providing an external parameter name with `_`. To further simplify the model, the shorthand `#` for specifying a parameter name has been removed, as have the special rules for default arguments. (17218256)

  Declaration

  ```
  func printFunction(str: String, newline: Bool)
  func printMethod(str: String, newline: Bool)
  func printFunctionOmitParameterName(str: String, _  newline: Bool)
  ```

Call

```
printFunction(“hello”, newline: true)
printMethod(“hello”, newline: true)
printFunctionOmitParameterName("hello", true)
```

- `NS_OPTIONS` types get imported as conforming to the `OptionSetType` protocol, which presents a set-like interface for options. (18069205)

  Instead of using bitwise operations such as:

  ```
  // Swift 1.2:
  object.invokeMethodWithOptions(.OptionA | .OptionB)
  object.invokeMethodWithOptions(nil)

  if options & .OptionC == .OptionC {
    // .OptionC is set
  }
  ```

  Option sets support set literal syntax, and set-like methods such as `contains`:

  ```
  object.invokeMethodWithOptions([.OptionA, .OptionB])
  object.invokeMethodWithOptions([])

  if options.contains(.OptionC) {
    // .OptionC is set
  }
  ```

  A new option set type can be written in Swift as a struct that conforms to the `OptionSetType` protocol. If the type specifies a `rawValue` property and option constants as `static let` constants, the standard library will provide default implementations of the rest of the option set API:

  ```
  struct MyOptions: OptionSetType {
    let rawValue: Int

    static let TuringMachine  = MyOptions(rawValue: 1)
    static let LambdaCalculus = MyOptions(rawValue: 2)
    static let VonNeumann     = MyOptions(rawValue: 4)
  }

let churchTuring: MyOptions = [.TuringMachine, .LambdaCalculus]
```

- The `do/while` loop is renamed to `repeat/while` to make it obvious whether a statement is a loop from its leading keyword. (20336424)

  In Swift 1.2:

  ```
  do {
  ...
  } while <condition>
  ```

  In Swift 2.0:

  ```
  repeat {
  ...
  } while <condition>
  ```

- `println` and `print` have been merged together into a single `print` function with a default argument (20775683)

  In Swift 1.2:
  ```
  func print(<stuff to print>)
  func println(<stuff to print>)
  ```

  In Swift 2.0:
  ```
  func print(<stuff to print>, appendNewline: Bool = true)
  ```

- Swift documentation comments use a syntax based on the Markdown format, aligning them with rich comments in playgrounds. (20180161)

  Outermost list items are interpreted as special fields and are highlighted in Xcode's QuickHelp.

  There are two methods of documenting parameters: parameter outlines and separate parameter fields. You can mix and match these forms as you see fit in any order or continuity throughout the doc comment. Because these are parsed as list items, you can nest arbitrary content underneath them.

  Parameter outline syntax:

  ```
  - Parameters:
    - x: ...
    - y: ...
  ```

  Separate parameter fields:

  ```
  - parameter x: ...
  - parameter y: ...
  ```

  Documenting return values:

  ```
  - returns: ...
  ```

  Other special fields are highlighted in QuickHelp, as well as rendering support for all of Markdown.

- The `CFunctionPointer<T -> U>` type has been removed; C function types are specified using the new `@convention(c)` attribute. Like other function types, `@convention(c) T -> U` is not nullable unless made optional. The `@objc_block` attribute for specifying block types has also been removed and replaced by `@convention(block)`. (16339559)

- Type annotations are no longer allowed in patterns and are considered part of the outlying declaration. (20167393)

  This means that code previously written as:

  ```
  var (a : Int, b : Float) = foo()
  ```

  needs to be written as:

  ```
  var (a,b) : (Int, Float) = foo()
  ```

  if an explicit type annotation is needed. The former syntax was ambiguous with tuple element labels.

- Deprecated enum elements no longer affect the names of non-deprecated elements when an Objective-C enum is imported into Swift. This may cause the Swift names of some enum elements to change. (17686122)

- All enums imported from C are `RawRepresentable`, including those not declared with `NS_ENUM` or `NS_OPTIONS`. As part of this change, the `value` property of such enums has been renamed `rawValue`. (18702016)

- `find` has been renamed to `indexOf()`. `sort` has been renamed to `sortInPlace()`, and `sorted()` becomes `sort()`.

- `String.toInt()` has been renamed to a failable `Int(String)` initializer, since initialization syntax is the preferred style for type conversions.

- `String` no longer conforms to `SequenceType`, to prevent non-unicode correct sequence algorithms from being prominently available on String. To perform grapheme-cluster-based, UTF8-based, or UTF-16-based algorithms, use the `.characters`, `.utf8`, and `.utf16` projections respectively.

- Generic functions that declare type parameters not used within the generic function's type produce a compiler error. For example:

```
func foo<T>() { } // error: generic parameter 'T' is not used in function
signature
```

Objective-C Language Features

- Lightweight generics allow you to specify type information for collection classes like `NSArray`, `NSSet`, and `NSDictionary`. The type information improves Swift access when you bridge from Objective-C and simplifies the code you have to write. (6294649)

  For example:

```
NSArray<UIImage *> *images;
NSDictionary<NSString *, NSURL *> *resourcesByName;
```

- A `kindof` type is introduced. Objects declared as `__kindof` types behave like a mix of 'id' and a specific object type: they specify an upper bound (e.g., must be `UIView` or a subclass thereof) but allow implicit downcasting to any subtype of that upper bound. (19589424)

  For example, if we assume that `-[UIView subviewWithTag:]` produces a kindof type `UIView *`, then:

```
UIButton *button = [view subviewWithTag:0]; // okay: UIButton is a UIView
[[view subviewWithTag:0] setTitle:@"Bounded" forState: UIControlStateNormal]; //
okay: method found in UIButton
UIResponder *responder = [view subviewWithTag:0]; // okay: UIView is a
UIResponder
NSString *string = [view subviewWithTag:0]; // error: UIView is unrelated to
NSString
```

- C functions that return Core Foundation objects via out-parameters can describe whether the object is returned at +0 or +1. (18742441)

```
OSStatus MyCFGetImportantValue(CFDictionaryRef data, CFStringRef __nullable *
__nonnull CF_RETURNS_NOT_RETAINED outImportantValue);
OSStatus MyCFCopyImportantValue(CFDictionaryRef data, CFStringRef __nullable *
__nonnull CF_RETURNS_RETAINED outImportantValue);
```

- The `NS_SWIFT_NAME` macro can be used to control the imports of enumerations whose constants don't map cleanly to Swift. (19240897)

  For example:

```
typedef NS_ENUM(NSInteger, DisplayMode) {
  DisplayMode256Colors NS_SWIFT_NAME(With256Colors),
  DisplayModeThousandsOfColors,
  DisplayModeMillionsOfColors
};
```

  is imported into Swift as

```
@objc enum DisplayMode : Int {
  case With256Colors
  case ThousandsOfColors
  case MillionsOfColors
}
```

  The macro can also be used to control whether factory methods are imported as initializers. For example,

```
@interface MyController : UIViewController
+ (instancetype)standardControllerForURLKind:(URLKind)kind
NS_SWIFT_NAME(init(URLKind:));
@end
```

  will be imported into Swift as

```
class MyController : UIViewController {
  init(URLKind kind: URLKind)
}
```

  even though its name does not follow the convention for automatic factory method importing.