Xcode Release Notes

About Xcode 6 beta 5

Supported Configurations

Xcode 6 requires a Mac running OS X 10.9.3 or OS X 10.10.

Xcode 6 includes SDKs for iOS 8 and OS X versions 10.9 and 10.10. To develop apps targeting prior versions of OS X or iOS, see the section “About SDKs and the iOS 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.

Installation

This release is a single application bundle. To install, double-click the downloaded DMG file, and drag the Xcode6-Beta5.app file to your Applications folder.

From within Xcode, you launch additional developer tools, such as Instruments and FileMerge, via the Xcode > Open Developer Tool command. 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 either OS X 10.9.4 with OS X Server 3.2 Developer Preview or OS X 10.10 beta with OS X Server 4.0 developer preview.

Once you have installed OS X, OS X Server and Xcode, follow these instructions to point OS X Server to this Xcode beta.

1. Open Server.app
2. Select the Xcode service
3. Choose Xcode

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 on 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.

New Features in Xcode 6

Swift Language

Swift is a new object-oriented programming language for iOS and OS X development. Swift is modern, powerful, expressive, and easy to use.

- Access all of the Cocoa and Cocoa Touch frameworks with Swift.
- Swift code is compiled and optimized by the advanced LLVM compiler to create high-performance apps.
- Safe by design: Swift pairs increased type safety with type inference, restricts direct access to pointers, and automatically manages memory using ARC to make it easy for you to use Swift and create secure, stable software. Other features related to language safety include mandatory variable initialization, automatic bounds checking to prevent overflows, conditionals break by default, and elimination of pointers to direct memory by default.
- Write, debug, and maintain less code, with an easy to write and read syntax and no headers to maintain.
- Swift includes optionals, generics, closures, tuples, and other modern language features. Inspired by and improving upon Objective-C, Swift code feels natural to read and write.
- Use Swift interactively to experiment with your ideas and see instant results.
- Swift is a complete replacement for both the C and Objective-C languages. Swift provides full object-oriented features, and includes low-level language primitives such as types, flow control, and operators.

Xcode 6 Features for Swift

Playgrounds. Playgrounds are an interactive development environment allowing you to experiment with Swift for prototyping, testing ideas, and so forth. Some uses for playgrounds include:

- Designing a new algorithm, watching its results every step of the way
- Experimenting with new API or trying out new Swift syntax
- Creating new tests and then verifying that they work before promoting them into your test suite

Learn in a playground. You can open select documentation in a playground to learn from the tutorial in a graphically rich, interactive environment.

Read-eval-print loop (REPL) in LLDB. The debugging console in Xcode includes an interactive version of the Swift language built right in. Use Swift syntax to evaluate and interact with your running app, or write new code to see how it works in a script-like environment. REPL is available from within the Xcode console or by using LLDB from within Terminal.

Per-language documentation. The Xcode documentation viewer shows Quick Help or reference documentation in the language of your choice—Objective-C, Swift, or both.

Synthesized interfaces. When using jump-to-definition for SDK content from Swift code, Xcode will synthesize a Swift view of the SDK API. This synthesized interface shows how the API is imported into Swift, and it retains all the comments from the original SDK headers.

Additional Feature Enhancements for Xcode 6 IDE

Testing

Performance measurement. The enhanced XCTest framework now supports the ability to quantify the performance of each part of an application. Xcode runs your performance tests and allows you to define a baseline performance metric. Each subsequent test run compares performance and displays the change over time.

Asynchronous code testing. XCTest now provides APIs for testing code that executes asynchronously. You can now create tests for network operations, file I/O, and other system interactions that execute using asynchronous calls.

Interface Builder

Live rendering. Interface Builder displays your custom objects at design time exactly as they appear when your app is run. When you update the code for your custom view, the Interface Builder design canvas updates automatically with the new look you just typed into the source editor, with no need to build and run.

Storyboards for OS X. Storyboards come to OS X with Xcode 6, taking advantage of the new view controller APIs in AppKit. Storyboards make it easy to wire together multiple views and define segue animations without writing code. Storyboards for OS X encourage interfaces that follow Mac standards so that your apps behave the way users expect.

Size classes. Size classes for iOS 8 enable designing a single universal storyboard with customized layouts for both iPhone and iPad. With size classes you can define common views and constraints once, and then add variations for each supported form factor. iOS Simulator and asset catalogs fully support size classes as well.

Custom iOS fonts. Interface Builder renders embedded custom fonts during design time, giving a more accurate preview of how the finished app will look, with correct dimensions.

Find and search. Interface Builder now supports find and search in `.xib` and `.storyboard` files.

Preview editor. The new preview editor includes the ability to present multiple previews and zooming.

Asset Catalogs

New support for image types. Size classes, JPEG, PDF, template images, and alignment rectangles are now supported by asset catalogs.

Debugger

View debugging. A single button click pauses your running app and “explodes” the paused UI into a 3D rendering, separating each layer of a stack of views. Using the view debugger makes it immediately obvious why an image may be clipped and invisible, and the order of the graphical elements becomes clear. By selecting any view, you can inspect the details by jumping to the relevant code in the assistant editor source view. The view debugger also displays Auto Layout constraints, making it easy to see where conflicts cause problems.

Enhanced queue debugging. The debug navigator records and displays recently executed blocks, as well as enqueued blocks. You can use it to see where your enqueued blocks are and to examine the details of what's been set up to execute.

Debug gauges. Debug gauges provide at-a-glance information about resource usage while debugging, calling the developer's attention to previously unknown problems.

- *I/O gauges.* Two new gauges, Network Activity and File Activity, visually highlight spikes in input/output activity while your app is running.
- *iCloud gauge.* Updated with support for the new Documents in the Cloud and CloudKit features that provide access to files outside the app-specific container.

GPU Tools

Metal support. Metal provides a new, low-overhead, GPU graphics and compute API as well as a shading language for iOS. The Metal shader compiler adds support for precompiling Metal shaders in Xcode. The GPU frame debugger and shader profiler supports debugging and profiling Metal-based games and apps.

Sprite Kit

Level designer. Support for Sprite Kit has been enhanced with a new Sprite Kit level designer and improved display of Sprite Kit variables when debugging.

Support for iOS. Sprite Kit and Scene Kit are now enhanced to work together and on iOS.

Extensions and Frameworks

Extensions support. You can add an extension target to any iOS or Mac app to expand your app's functionality to other apps in the OS.

Frameworks for iOS. iOS developers can now create dynamic frameworks.

iOS Simulator

Configurations. New iOS Simulator configurations allow you to keep data and configuration settings grouped together. Run one configuration for one version of an app, with its own data, and another configuration for a different app version.

Localization

XLIFF import-export. Xcode can package your localizable strings into the industry standard XLIFF format to send off for localization.

Implicit .strings file. Xcode automatically generates the base language .strings file directly from your source code.

Preview in Interface Builder. While designing in Interface Builder, the preview assistant can show how the interface appears in other languages.

Run in locale. Xcode can run your app in the iOS Simulator, or directly on devices, as it would appear to customers in other countries.

Compiler

Profile Guided Optimization. Profile Guided Optimization (PGO) works with the LLVM optimizer and XCTest tests to profile the most actively used parts of your application. You can also exercise your app manually to generate an optimization profile. PGO uses the profile to further optimize your app, targeting the areas that most need optimization, improving performance beyond what setting optimization options alone can achieve.

User-defined modules. Developers are now able to define modules for their own Objective-C code, making it easier than ever for them to share frameworks across all their projects.

Instruments

New user interface. The new Instruments user interface makes configuring your performance tuning session easier and improves control. The new template chooser allows you to choose your device and target as well as the starting point for your profiling session. The track view allows direct click-and-drag to set the time filter range. The toolbar takes up less space to let you focus on the task at hand. The tracks of recorded data are given more space, and configuration for how data is collected and viewed is managed in a unified inspector area.

Profile tests. You can choose any test or test suite to profile, which is useful for analyzing memory leaks in a functional test or time profiling a performance test to see why it has regressed.

Support for simulator configurations. Simulator configurations are treated like devices by Instruments, making it easy to launch or attach to processes in the simulator.

New Counters instrument. Counters and Events instruments have been combined into a more powerful instrument and made easier to configure. It can track individual CPU events, and you can specify formulas to measure event aggregates, ratios, and more. iOS developers on 64-bit devices can now use Counters to fine-tune apps.

Swift and Extensions support. Of course, Swift is supported—you'll see Swift symbols in stack traces and Swift types in Allocations. You can also use Instruments to profile your app extensions.

Xcode Server

Triggers. Triggers allow you to make more complex integration scenarios by configuring server-side rules to launch custom scripts before or after the execution of an Xcode scheme.

Performance test integrations. Xcode Server supports the new Xcode performance-testing features, making it easy for a team to share a group of devices and Macs for continual performance testing.

Delta tracking. Issues are now tracked per integration, so you can see when an issue appeared or when it or was fixed, and by whom.

Greater control. Configuration options in Xcode Server give development teams even greater control over the execution of bots. New settings for integration intervals, grouping of bots, and iOS Simulator configurations make Xcode bots more powerful than ever. The new reports UI includes bot-level statistics, the number of successful integrations, as well as commit and test addition tracking.

New in Xcode 6 beta 5

Refinements to Optional Types in Swift

- The `UIView`, `NSView`, `UIFont`, and `UIApplicationDelegate` classes have been audited for optional conformance, removing most implicitly unwrapped optionals from their interfaces. This clarifies the nullability of their properties and arguments / return values of their methods. The changes in Beta 5 are simply the first step; more refinements to the SDK will come in future betas. (17892713)

- Optionals can now be compared to `nil` with == and `!=`, even if the underlying element is not `Equatable`. (17489239)

- Optionals no longer conform to the `BooleanType` (formerly `LogicValue`) protocol, so they may no longer be used in place of boolean expressions (they must be explicitly compared with `v != nil`). This resolves confusion around `Bool?` and related types, makes code more explicit about what test is expected, and is more consistent with the rest of the language.

  Note that `ImplicitlyUnwrappedOptional` still includes some `BooleanType` functionality. This issue will be resolved in a future beta. (17110911)

- The optional unwrapping operator `x!` can now be assigned through. Mutating methods and operators can be applied through it. (16922562)

  For example:

```
var x: Int! = 0
x! = 2
x!++

// Nested dictionaries can now be mutated directly:
var sequences = ["fibonacci": [1, 1, 2, 3, 0]]
sequences["fibonacci"]![4] = 5
sequences["fibonacci"]!.append(8)
```

  The `?` chaining operator can also be used to conditionally assign through an optional if it has a value:

```
sequences["fibonacci"]?.append(0)
sequences["fibonacci"]?[6] = 13
sequences["perfect"]?[0] = 6
       // Does nothing because the sequence has no 'perfect' key
```

  Note that properties that are optional as a result of lookup through `AnyObject` cannot be mutated yet.

- A new `??` nil coalescing operator has been introduced. `??` is a short-circuiting operator, similar to `&&` and `||`, which takes an optional on the left and a lazily-evaluated non-optional expression on the right. (15247356)

For example:

```
public func ?? <T> (optional: T?, defaultValue: @autoclosure () -> T) -> T {
     switch optional {
           case .Some(let value):
                 return value
           case .None:
                 return defaultValue()
     }
}
```

The nil coalescing operator provides commonly useful behavior when working with optionals, and codifies this operation with a standardized name. If the optional has a value, its value is returned as a non-optional; otherwise, the expression on the right is evaluated and returned.

```
var myArray: [Int] = []
print(myArray.first ?? 0) // produces 0, because myArray.first is nil
myArray.append(22)
print(myArray.first ?? 0) // produces 22, the value of myArray.first
```

Ranges in Swift

The idea of a Range has been split into three separate concepts:

- Ranges, which are Collections of consecutive discrete `ForwardIndexType` values. Ranges are used for slicing and iteration.
- Intervals over `Comparable` values, which can efficiently check for containment. Intervals are used for pattern matching in switch statements and by the ~= operator.
- Striding over `Strideable` values, which are `Comparable` and can be advanced an arbitrary distance in O(1).

Some of the types most commonly used with the range operators `..<` and `...`—for example, `Int`—conform to both `Comparable` and `ForwardIndexType`. When used in a context that requires pattern matching (such as a switch case), the range operators create *Intervals*. Otherwise they create *Ranges*. Therefore, in a context without type constraint such as `let x = 3..<10`, the result is a *Range*.

It is considered an error to form a *Range* whose `endIndex` is not reachable from its `startIndex` by incrementation, or an *Interval* whose end is less than its start. In these cases, *Interval* formation always traps and *Range* formation traps when a violation is detectable, that is, when the indices are `Comparable`.

```
1> 1…0
fatal error: Can't form Range with end < start
```

Intervals are represented by two generic types, `HalfOpenInterval<T>`, created by the `..<` operator, and `ClosedInterval<T>`, created by the `...` operator:

```
1> 3.14..<12
$R0: HalfOpenInterval<Double> = {
  _start = 3.1400000000000001
  _end = 12
}
2> 22...99.1
```

```
$R1: ClosedInterval<Double> = {
_start = 22
_end = 99.099999999999994
}
```

A *range* `x..<y` always has `startIndex == x`. Therefore, `x` is the first valid subscript, and this applies even when the `Index` type is `Int`. In other words, the first valid subscript of `5..<10 is 5`, not `0`. To prevent surprise, it is a compilation error to subscript a range over an `Integer` type outside a generic context (for example, expressions like `(5..<10)[0]`).

All *Range*s are represented by instances of a single generic type, `Range<T>`, whose representation is always half-open (and thus always print in the REPL and Playgrounds as a half-open range). Currently an inclusive range cannot include the last value in a sequence (for example, `4…Int.max` doesn't work) unless the context requires an *Interval* (like a case pattern matching specification).

Addition changes to `Range`:

- `ReverseRange<T>` has been removed; use `lazy(x..<y).reverse()` instead.
- New functions have been added that return a `Sequence` beginning with `start` and continuing up to `end`, or through `end`, respectively, by repeatedly adding step:

```
// stride<T:Strideable>(from start: T, to end: T, by step: T.Distance)
1> for x in stride(from: 3.1, to: 3.14, by: 0.02) { println(x) }
3.1
3.12

// stride<T:Strideable>(from start: T, through end: T, by step:
T.Distance)
2> for x in stride(from: 3.1, through: 3.14, by: 0.02) { println(x) }
3.1
3.12
3.14
```

- The Range's `.by(d)` method is removed. `Use stride(from: to: by:)` or `stride(from: through: by:)` instead. (17220839)

Swift Standard Library

- Several protocols (for example, the Integer protocol) in the standard library are renamed to reduce confusion. The standard library protocols aim to be named with a suffix of `ible`, `able`, or `Type`. Along with this change, the `LogicValue` protocol has been renamed to `BooleanType` and its `getLogicValue()` method has been changed to a `boolValue` property. (17165920)
- The `UnsafeConstPointer` and `UnsafePointer` types have been renamed to `UnsafePointer` and `UnsafeMutablePointer` for consistency with Cocoa and to encourage immutability. The Autoreleasing and other pointers have been renamed for consistency as well. (17892766)

- The `UnsafeArray` and `UnsafeMutableArray` types (which correspond to the C "pointer + length" memory buffer concept, but without ownership guarantees) have been renamed to `UnsafeBufferPointer` and `UnsafeMutableBufferPointer` to emphasize their reference semantic behavior and distinguish them from language arrays. (17892775)

- Array provides `first` and `last` properties that return the corresponding element or `nil` if empty. (17768202)

`Dynamic` declaration modifier

The new `dynamic` declaration modifier has been introduced to elevate privileged dynamic dispatch to a first class language concept which is orthogonal from the `@objc` attribute. When applied to a method, property, subscript, or initializer, it guarantees that references to the declaration are always dynamically dispatched, never inlined or devirtualized, and that the method binding can be reliably changed at runtime.

This change also clarifies the `@objc` attribute. Now it only makes a declaration visible to Objective-C (for examples, to `objc_msgSend`), instead of conflating exposure to the Objective-C runtime with guaranteed lack of devirtualization. This is a cleaner conceptual semantic model that enables performance improvements for a wide range of NSObject subclasses by allowing the compiler to use more efficient dispatch and access techniques for declarations marked `@objc` (which is usually implicit).

```
final class Foo : SomeBaseClass {
     // Always accessed by objc_msgSend: may be swizzled at runtime, etc,
     // even though Foo is final.
     dynamic var x: Int

      // Visible and accessed by objc_msgSend from ObjC;
      // may be accessed by more efficient lookup mechanisms in Swift.
      @objc var y: Int

      // Exposed to ObjC if SomeBaseClass is an Objective-C class,
      // or derived from one.
      var z: Int
 }
```

The `dynamic` keyword enables KVO, proxying, and other advanced Cocoa features to work reliably with Swift declarations. Though the feature is independent of the `@objc` attribute, the implementation still currently relies on Objective-C runtime details, so `dynamic` currently can only be applied to declarations with `@objc`-compatible types. (17540067)

Swift Operators and Attributes

- The `@auto_closure` attribute has been renamed to `@autoclosure`. (16859927)

- The `@assignment` attribute has been removed from operator implementations. (16656024)

- The `@prefix`, `@infix`, and `@postfix` attributes have been changed to declaration modifiers, so they are no longer spelled with an `@` sign (`now, prefix func (…)`). Operator declarations

have been rearranged from `operator prefix - {}` to `prefix operator - {}` for consistency. (17527000)

- The `@class_protocol` attribute has been removed; the new syntax for declaring that only protocol conformance is limited to classes is 'protocol P : class { … }'. (17510790)

- The += operator on arrays only concatenates arrays, it does not append an element. This resolves ambiguity working with `Any`, `AnyObject` and related types. (17151420)

Swift Command Line Interface

- The swift compiler executable has been renamed to `swiftc` and a new `swift` executable is available specifically to enable natural `#!` scripts (replacing the former "-i" mode). On the command line, `xcrun swift myscript.swift` runs in immediate mode by default. (17710788)

- The `-Ofast` compiler option, which produces the fastest code by disabling integer overflow, array bounds checks, and other safety checks, is being renamed to `-Ounchecked`. In this beta, both options are accepted, but `-Ofast` will disappear in a future beta. (17202004)

Miscellaneous Swift

- When force-casting between arrays of class or `@objc` protocol types using `a as [C]`, type checking is deferred until the moment each element is accessed. Because bridging conversions from `NSArray` are equivalent to force-casts from `[NSArray]`, this makes certain `Array` round-trips through Objective-C code O(1) instead of O(N). (17340393)

- Swift recognizes comments that start with the special marker `/**` or `///` as documentation comments. Swift uses `reStructuredText`, an easy-to-read, modern lightweight markup language in documentation comments. Currently only block-level markup is supported (nested bullet and enumerated lists, field lists). In order to document function and method parameters, use `:param:` followed by parameter name, and to document return values use `:returns:`. (17839919)

- Swift performance has been improved for code with loops, code with complex control flow and for Debug builds. Further improvements will come in a future beta. (17811631)

- Overrides of a designated initializer must be marked with the 'override' modifier. (17892845)

- The `required` modifier is written before every subclass implementation of a required initializer. Required initializers can be satisfied by automatically inherited initializers. (17892840)

Xcode support for Swift

- The Swift optimization level of `-Ofast` has been deprecated and removed. Projects should be updated to use `-Ounchecked` instead. (17795215)

- Xcode supports brace and paren matching and type-over completions for Swift code. (15191101)

- Xcode properly handles single-file actions (e.g. compile, analyze, pre-process, assemble) for Swift files. (14978271)

Swift REPL Breakpoints

Breakpoints can be set in the Swift REPL. Any functions or methods defined in REPL source can be executed on demand, so it's useful to be able to set breakpoints in them. (15580623)

The REPL source is considered the current source file when at the REPL prompt, so there's no need to specify a source file name, just the line number as shown in the following example:

```
1> func test() {
2.    println("This is a test")
3. }
4> :b 2
5> test()
```

When stopped at a breakpoint the LLDB prompt replaces the REPL prompt until the last expression (the call to test on line 5 in our example) completes or results in a fatal error.

Playgrounds

- Calls to `print()` and `println()` in playgrounds reflect their output as results associated with the source line in addition to showing the full console output in the timeline assistant. (16762680)

- Playgrounds support importing frameworks. This provides a way to share code between your applications and playgrounds, which can both import your frameworks. To do this, your playground must be in the same workspace as the project that produces your framework. You must have already built your framework. If it is an iOS framework, it must be built for a 64-bit run destination (e.g. iPhone 5s). You must have an active scheme which builds at least one target (that target's build location will be used in the framework search path for the playground). Your "Build Location" preference (in advanced "Locations" settings) should not be set to "Legacy". If your framework is not a Swift framework the "Defines Module" build setting must be set to "Yes". Once all these conditions are fulfilled, importing your framework works in a playground. (17576393)

- If your playground code crashes or throws an uncaught exception Xcode shows the line that crashed in the playground editor. Click on Quick Look to show the backtrace in the results sidebar for that line. (16637303)

Instruments

- The Automation instrument requires enabling the "UI Automation" setting on the targeted iOS device. (16732303)

Xcode Server

- Xcode Server prunes older integration data based on available disk space. (16535845)
- Commits for an integration appear in a separate tab in the integration report. Changes for each commit are available by clicking on the revision number. (16676762)
- When looking at an integration, the product (`.ipa` or `.pkg`) and archive produced by that integration are available for download. (16676815)
- Canceling an integration cancels any triggers running as part of that integration. (17578403)
- Email notifications from Xcode Server include a link to open the integration in Xcode (17616704)

General

- When finding text in the Find navigator using a regular expression, the `\1` syntax is no longer supported in the replacement string. To refer to a captured group, use the syntax `$123`. With this change, you can insert a digit after the tenth captured group and beyond by escaping the digit, e.g. “$10\2”. Likewise, when finding text using patterns, you can insert a digit after the tenth pattern. (11836632)
- Tools that support Carbon development are deprecated with Xcode 6. These include: `BuildStrings`, `GetFileInfo`, `SplitForks`, `ResMerger`, `UnRezWack`, `MergePef`, `RezWack`, `SetFile`, `RezDet`, `CpMac`, `DeRez`, `MvMac`, and `Rez`. (10344338)
- The Printer Simulator has been moved to the Hardware IO Tools package. (17014426)

Issues Resolved in Xcode 6 beta 5

Playgrounds

- iOS playground execution is sandboxed. (17437417)

iOS Simulator

- It is safe to click outside the app window in the resizable iPhone Simulator. (17022609)

Swift Language

- A `CGFloat` can be constructed from any `Integer` type (including the sized integer types) and vice-versa. (17670817)

  Also, `CGFloat` allows striding, for example:

  ```
  for f: CGFloat in stride(from: 1.0, through: 2.0, by: 0.5) {
      println(f)
  }
  ```

- String comparison operators, such as ==, <, <=, were fixed to perform locale-insensitive comparison according to the Unicode collation algorithm. (17498444)

- String had a `compare()` method that conflicted with the `NSString` `compare()` API reflected when importing Foundation. The former has been removed to avoid ambiguity with the latter. (17800504)

Swift REPL

- Structs defined in the Swift REPL can be constructed using their implicit constructors. (17670241)

Xcode Server

- Bots configured to create archives for Mac are again able to do so. (17700710)

- Issues that were marked as resolved in a previous integration and new in the next are no longer be displayed incorrectly. (17412053)

- Version information for your Xcode Server appears correctly in the accounts panel in Xcode 6. (17510821)

- The Next button is no longer disabled when creating or editing a bot. (17683360)

- When configuring repository authentication for a bot, choosing New SSH Key produces an SSH key that functions correctly when the public key is installed on a remote server. (17755235)

- Bots can run against "All iOS Simulators." (17559029)

Known Issues in Xcode 6 beta 5

Swift

- Weak linking against frameworks that are new to iOS 8 or OS X 10.10 and running on older versions of these operating systems does not work. (17825012)

- OS X apps using SpriteKit and Swift may crash on OS X Mavericks. The crashes occur in `objc_release` when the autorelease pool is popped. (17887502)

  Workaround: Convert Swift strings to `NSString` explicitly before using them with the `imageNamed:` or `fontNamed:` initializers:

  ```
  let node = SKSpriteNode(imageNamed: NSString(string: "foo.png"))
  ```

- Some built command line tools have a dependency on dynamic libraries inside Xcode and are able to be run if Xcode is installed in the same location on disk as it was when the tool was built. (16866827)

- Unit tests written in Objective-C cannot currently import the Swift generated interfaces header (that is, `$(PRODUCT_MODULE_NAME)-Swift.h`) for application targets, and therefore cannot be used to test code that requires this header. (16931027)

  Workaround: Unit tests for Swift code should be written in Swift. Unit tests written in Objective-C for framework targets can access the Swift generated interfaces by importing the framework module using `@import FrameworkName;`.

- Xcode does not support building static libraries that include Swift code. (17181019)

- If you set a Swift subclass of `NSValueTransformer` as a binding's value transformer, the Xib or storyboard contains an invalid reference to the class and the binding does not work properly at runtime. (17495784)

  Workaround: Either enter a mangled class name into the Value Transformer field or add the `@objc(…)` attribute to the `NSValueTransformer` subclass.

- The refactoring engine does not detect when an Objective-C class has a Swift subclass. (16465974)

  Workaround: When doing Objective-C refactoring, any changes needed in Swift subclasses need to be made by hand.

- ARM and ARM64 code that uses the `float16_t` type may fail when trying to link C++ code compiled with an older compiler. In previous versions of the compiler, `float16_t` was defined as `uint16_t`; `float16_t` is defined as `__fp16`. (15506420)

- Custom CloudKit container identifiers must be prefixed with "iCloud." when added through Xcode. (17028017)

- After porting a custom class from Objective-C to Swift, any references to the class in a Xib or storyboard need to be updated manually. (17153630)

    Workaround: Select each reference, then in the Custom Class inspector, clear the Class field, save, and reenter the class name.

App Extensions

- Debugging Keyboard Extensions is not supported. (16879317)

- Debugging Finder Sync is not supported. (17849671)

- Developers with existing Finder Sync App Extension templates need to update their code to work with this release. (16810473)

    1. Change the extension to inherit from FIFinderSync rather than just implement an FIFinderSync interface
    2. In the Info.plist, set:
    ```
    PrincipalClass = NSApplication
    LSUIElement = YES
    ```

- Signed OS X app extensions may emit dyld warnings and fail to launch. (17711503)

    Workaround: Ensure that the same team ID is set for both the app extension and the containing app.

- Debugging for Document Provider and Document Picker for iOS may not work. (16567805)

- Using `xpc_service_set_attach_handler` to profile App Extensions for keyboard isn't working. (16946100)

    Workaround: Trigger the Keyboard extension to load and then attach to the running instance (for example, `com.thirdparty.foo.keyboard (123)`) via Instruments's target chooser under the running simulator device.

- Creating a Document Picker extension with the iOS File Provider enabled does not add the containing app to the resulting app group. (16871267)

    Workaround: Add the containing app to the app group manually.

- iOS Extension debugging often doesn't work after the first time the extension container app is installed on the device. For example in the Today view the widget may not appear or it may not launch. (17690419)

    Workaround: Deleting the container app from the device or toggling the enable/disable state for the extension can cause it to be loaded. Xcode should then attach to it normally.

- When you create an extension project from the templates in Xcode the `NSExtension/NSActivationRule` is set to filter for all types expect photos. Due to this, new action and sharing extension apps are showing up in many places. (17818496)

    Workaround: Update the `NSExtension/NSActivationRule` property in the `Info.plist` of the extension to control where your extension app is made available on iOS. Please refer to the App Extension Programming Guide for more details on `NSExtensionActivationRule`.

    Note: you must use an appropriate `NSExtensionActivationRule` when submitting to the App Store.

Interface Builder

- A storyboard may fail to compile after adding an `NSCollectionView` to it. (17009377)

    Workaround: Pick a xib that includes the `NSCollectionView` and load it into a Storyboard based View.

- Interface Builder does not support connecting to an outlet in a Swift file when the outlet's type is a protocol. (17023935)

    Workaround: Declare the outlet's type as `AnyObject` or `NSObject`, connect objects to the outlet using Interface Builder, then change the outlet's type back to the protocol.

iOS Simulator

- Location functionality may not work in the Simulator. (17858614)

- Renaming Xcode after running any of the Xcode tools in that bundle may cause the iOS simulator to be no longer available. (16646772)

    Workaround: Either rename Xcode back to what it was when first launched or reboot your Mac.

- Weak linking against frameworks that are new to iOS 8 and running on iOS 7.1 may not work in the iOS Simulator if the host is OS X Yosemite. (17807439)

App Validation

- Validating archives may not work. (16891311)

Xcode Server

- Older betas of Xcode 6 Server are incompatible with this release of Xcode 6. Xcode 6 betas only supports connecting to OS X Server running the same or later beta versions of Xcode 6 beta. Upgrade and migration between beta builds of Xcode Server is not yet supported. (17732847)

- When the Xcode service is disabled, bots continue to integrate. (17633884)
- After setting the development bit on a device, bots on simulators may stop working. (17873064)

    Workaround: Disconnect and reconnect the device.

New in Xcode 6 beta 4

Swift Enables Access Control

Swift access control has three access levels:
• `private` entities can only be accessed from within the source file where they are defined.
• `internal` entities can be accessed anywhere within the target where they are defined.
• `public` entities can be accessed from anywhere within the target and from any other context that imports the current target's module.

By default, most entities in a source file have internal access. This allows application developers to largely ignore access control while allowing framework developers full control over a framework's API.

Applications do not require any changes in almost all cases. Frameworks need the public API marked as public. Note that implicitly-synthesized initializers for classes and structs are internal by default; to expose these publicly, write them explicitly.

```
// An example class in a framework target.
public class ListItem: NSObject {
    public var text: String
    public var isComplete: Bool

    // Readable throughout the module, but only writeable from
    // within this file.
    private(set) var UUID: NSUUID

    public init(text: String, completed: Bool, UUID: NSUUID) {
        self.text = text
        self.isComplete = completed
        self.UUID = UUID
    }

    func refreshIdentity() {
        self.UUID = NSUUID()
    }

    // Must be public because it overrides a public method and is
itself
    // part of a public type.
    public override func isEqual(object: AnyObject?) -> Bool {
        if let item = object as? ListItem {
            return self.UUID == item.UUID
        }
        return false
    }
}
```

Because the generated header for a framework is part of the framework's public Objective-C interface, only declarations marked public appear in the generated header for a framework. For applications, the generated header contains both public and internal declarations.

Declarations marked private are not exposed to the Objective-C runtime if not otherwise annotated. IB outlets, IB actions, and Core Data managed properties remain exposed to Objective-C whatever their access level. If you need a private method or property to be callable from Objective-C (such as for an older API that uses a selector-based callback), add the `@objc` attribute to the declaration explicitly.

A limitation of the access control system is that unit tests cannot interact with the classes and methods in an application unless they are marked public. This is because the unit test target is not part of the application module.

For more information, read The Swift Programming Language and Using Swift with Cocoa and Objective-C. (15747445)

New stride() Functions

The `.by()` method for ranges has been replaced with general `stride()` functions. To adopt `stride()`, use `stride(from: to: by:)` for exclusive ranges and `stride(from: through: by:)` for inclusive ranges. (17668462)

For example:

```
stride(from: x, to: y, by: z)          // was: (x..<y).by(z)
stride(from: x, through: y, by: z)     // was: (x...y).by(z)
```

More improvements are due in forthcoming betas, addressing a variety of issues iterating over floating point ranges, constructing negative ranges, and several other known range-related problems.

iOS 7 and OS X 10.9 minimum deployment target

The Swift compiler and Xcode enforce a minimum deployment target of iOS 7 or OS X Mavericks. Setting an earlier deployment target results in a build failure.

Revised Declaration Modifiers

The `@final`, `@lazy`, `@optional`, and `@required` attributes have been converted to declaration modifiers, specified without an @ sign. Future betas will include improvements to `@class_protocol` and adjust `@prefix` and other operator attributes. (17168115)

Landmarks

Xcode supports `//MARK:`, `//TODO:` and `//FIXME` landmarks to annotate your code and lists them in the jump bar. (14768427)

Unicode String improvements

The `String` type implements a grapheme cluster segmentation algorithm to produce `Characters`. This means that iteration over complex strings that include combining marks, variation sequences, and regional indicators work properly. For example, this code returns the value `15: countElements("a\u{1F30D}cafe\u{0301}umbrella\u{FE0E}\u{1F1E9}\u{1F1EA}”)`

Also, a `for–in` loop over the string produces each human visible character in sequence. (16013860)

Source Control Logging

When viewing logs for a bot, source control logging is now broken out into its own log and indicates what will be and was checked out for each integration. (17715180)

Xcode Server

- When testing is disabled for a bot, users will now be given the option for which platform they wish to build. (16839464)
- Xcode Server now utilizes all available cores of your server. (16963617)

Issues Resolved in Xcode 6 beta 4

Swift

- The complete Swift language is now supported in the Swift REPL. Notable improvements include typealiases and operator declarations. (17730356)

- Support for editing multi-line declarations in the Swift REPL has been improved. (17730496) These include:
    - New lines insertion
    - Line break deletion
    - REPL history preserved across sessions (available by pressing up arrow)

- `NSUInteger` maps to Swift's `Int` type in iOS and OS X system frameworks and to `UInt` in user code . (17473606)

- `Process.arguments` reports arguments correctly when used in interactive mode (that is, `swift -i`). (17191889)

- CGFloat is now a distinct floating-point type that wraps either a Float on 32-bit architectures or a Double on 64-bit architectures. It provide all of the same comparison and arithmetic operations of Float and Double and may be created using numeric literals. Using CGFloat insulates your code from situations where your code would be fine for 32-bit but fail when building for 64-bit or vice versa. (17224725)

- The `CString` type has been removed. Values of type `const char *` are now imported as `ConstUnsafePointer<Int8>` instead of `CString.` C macros that expand to string literals are imported as `String`.

- The `modulusWithOverflow` static method on integer types was renamed to `remainderWithOverflow`.

- The `\x`, `\u` and `\U` escape sequences in string literals have been consolidated into a single and less error prone `\u{123456}` syntax. (17279286)

- The `BooleanLiteralConvertible` protocol allows user-defined types to support Boolean literals. `true` and `false` are now Boolean literals and language keywords rather than library-defined constants. (17405310)

- Type inference for `for..in` loops has been improved to consider the sequence along with the element pattern. (16773836)

  For example, Swift now accepts the following loop that was previously rejected:
  `for i: Int8 in 0..<10 { }`

Playgrounds

- Playgrounds now only support loading resources via `NSBundle` and related API if they are stored inside of the playground document; the “Absolute Path” and “Relative to Playground” resource path options are no longer supported. (Other files stored in ~/Documents/Shared Playground Data are also accessible using standard path- or URL-based file loading API.) (17510954)

- OS X playgrounds now execute inside of a sandbox. The sandbox used for playground execution permits network access, but prohibits general access to the filesystem.

  To make sharing data between playgrounds possible, all playgrounds have read/write access to the folder "~/Documents/Shared Playground Data". Playgrounds can access this folder using the XCPSharedDataDirectoryPath constant in the XCPlayground framework. (Note that, for OS X playgrounds, this constant resolves to the path inside of the playground's container which is symlinked to the real "~/Documents/Shared Playground Data".)

  A playground will be given a new container every time it is opened. To ensure that data written out by an execution of a playground is accessible in the future, write it to "~/Documents/Shared Playground Data". (16773467)

Testing

- Profiling unit or performance tests in Instruments now works reliably when Instruments is in use. (17034363)

Interface Builder

- In Swift files, properties with the `@IBInspectable` attribute and Swift-only types such as `String` are now available. (17558064)

- Xibs that reference Carbon objects or xibs that load Interface Builder 3 plug-ins no longer cause a build error indicating a missing property list. (17467916)

- When using Size Classes, the size of the abstract editing canvas has increased by 25%. Existing projects will be automatically updated when opened. (17629007)

App Extensions

- Newly created Share and Action extensions will not automatically appear everywhere. Developers should configure Activation Rules to specify when these extensions should appear. (17690383)

Important Changes, Issues Resolved in Xcode 6 beta 1–3

Swift Language

- The online *Swift Programming Language* documentation and book have been updated. See: https://itunes.apple.com/us/book/the-swift-programming-language/id881256329?mt=11

- Array in Swift has been completely redesigned to have full value semantics like Dictionary and String have always had in Swift. This resolves various mutability problems – now a 'let' array is completely immutable, and a 'var' array is completely mutable – composes properly with Dictionary and String, and solves other deeper problems. Value semantics may be surprising if you are used to NSArray or C arrays: a copy of the array now produces a full and independent copy of all of the elements using an efficient lazy copy implementation. This is a major change for Array, and there are still some performance issues to be addressed. Please see the *Swift Programming Language* for more information. (17192555)

- The Array and Dictionary "sugar" syntax has been redesigned: You now declare arrays as `[Int]` instead of as `Int[]`, as shorthand for `Array<Int>`. The old syntax made sense when arrays had semantics closer to C arrays, but would be misleading with the new value semantics approach. Along with this, Dictionary syntax has improved so that `[Key:Value]` is treated as sugar for `Dictionary<Key,Value>`. Both of these are now consistent with each other and with the literal syntax used to build an array or dictionary. Please see the *Swift Programming Language* for more information.

- `NSDictionary*` is now imported from Objective-C APIs as `[NSObject : AnyObject]`. (16870626)

- The half-closed range operator has been changed from `..` to `..<` to reduce confusion and ambiguity. Now the two range operators are `..<` and `...` for half-closed and closed ranges, respectively (17203527).

- `nil` is now a literal in the language, not a global constant of `_Nil` type. This change resolved a number of problems with nil; e.g. nil in a collection, nil converting to `Any`, etc. Types can now indicate that they are nil compatible by conforming to the `NilLiteralConvertible` protocol. (16951729)

- APIs imported from C no longer use type aliases like `CInt` or `CFloat`, which obfuscated the underlying type. They are now imported as `Int32` and `Float`, etc.

- APIs imported from C that use C pointers are now imported with a much simpler API type structure which is more predictable, preserves `const` mutability in more cases, and preserves `__autoreleased` pointer information. Now you will see `UnsafePointer`, `ConstUnsafePointer`, `AutoreleasingUnsafePointer`, etc. Function pointers are also imported now, and can be referenced and passed around. However, you cannot call a C function pointer or convert a closure to C function pointer type.

- All APIs deprecated in iOS 7 or earlier and OS X 10.9 or earlier are now unavailable to use in Swift.

- The standard library improved in various ways:
  - The `succ()` and `pred()` methods in the standard library have been renamed to `successor()` and `predecessor()`.
  - Integer types now provide methods like `Int.addWithOverflow` which perform an operation, like add, and return the partial result along with a bit indicating whether overflow occurred.
  - Integer types now have a `byteSwapped` property to perform an unconditional byte swap, have an `init(bigEndian:)` / `init(littleEndian:)` member to perform a conversion from an integer of known endianness, and have bigEndian/littleEndian properties to project the integer value into a representation with a specific endianness.
  - The global `sort` function now mutates its first argument, and a new `sorted` function always returns a new collection.
- You can now declare an outlet's type to be a class that is implemented in Swift and connect it to an object in an Interface Builder document. You can also declare outlets in a Swift class and connect them to an instance of one of its subclasses in an Interface Builder document. (16968022)
- `@IBOutlets` may be explicitly marked `strong` to override their implicitly-weak behavior. (16954464)
- `unowned` class references will no longer sometimes retain their target. (16980445)
- Objects of a class type, such as NSObject or NSArray, can now be downcast to bridged Swift types. For example, given an NSArray named `nsarr`, one can write:

```
if let stringArr = nsarr as? [String] {
   // stringArr is an array of Strings
 }
```

  without having to first cast `nsarr` to an `AnyObject`. (16972956)

Playgrounds

- Playgrounds that use resources via the resource path setting in the file inspector will no longer eventually degrade Launch Services functionality. (17089171)
- Interactive Learning documents now display correctly when line wrapping is disabled. (17033148)
- The XCPlayground framework is now available for iOS playgrounds in this release. (17033128)

- In iOS playgrounds, `UIView` and subclasses now show correct QuickLook data in the sidebar even if the view uses Core Image filters, OpenGL layers or non-affine transforms to draw itself. (17029335)

- If a playground document causes Xcode to crash, you now have the option to avoid reopening the document. (16833321)

Swift REPL

- Global variable declarations now behave correctly the following circumstances:
    - Assigning from a tuple to multiple global variable declarations no longer results in uninitialized variables (for example: `var (x,y) = (10, "Hello")`).
    - Global variables declared in the REPL can be accessed from local scopes.
    - The expression defining the initial value of a global variable declaration is evaluated correctly in the REPL.

- The REPL allows functions and types to be redefined, but attempting to redefine a global variables no longer silently fails. Global constant declarations using the `let` keyword are currently treated as variables. (17033419)

General

- Code completion is now supported in Swift files and playgrounds that contain non-ASCII characters. (17067748)

- When exporting an XLIFF file to a file path containing a space an error will no longer be returned and the export completes. (17043515)

- Enabling a capability—for example, iCloud—in the target editor will no longer fails if the disclosure triangle is collapsed. (17030030)

- After adding iCloud documents and then adding CloudKit services for the first time, when you run the app you will no longer get an error stating (17033676):

  ```
  The executable was signed with invalid entitlements.
  ```

- When using the Devices window to take a screenshot from a device, the screenshot is now saved. (17057626)

- A new iOS device will now appear as enabled-for-development even before Xcode has pushed something onto that device. (16222862)

- Inspectors in the Utility area now indicate when they are collapsed. (14308392)

- xcodebuild now supports the "id" option for iOS simulator destination specifiers. In Xcode 5 this option was only available for iOS device destination specifiers. (17398965)

Debugger

- Debugging Keyboard Extensions is now supported in this release. (16879317)
- Debugging Today app extensions in the iOS Simulator is now supported in this release. (16947459)
- When invoking build and run with an app extension for the first time, Xcode will now attach. (17084882)
- When debugging app extensions on iOS Simulator, you will no longer see twice as many extensions waiting in the Debug Navigator. (16983273)
- If a project has not been updated to use LLDB, Xcode will not crash when you try to run your application. (16825484)

Editing User Interfaces

- Views that opt in to Interface Builder live rendering are now supported from within your application and no longer require a framework.
- Cross-view constraints on OS X are now properly preserved at runtime. (15622565)
- In an OS X `.xib` or `.storyboard` file, if you insert a segmented control from the Objects library or if you copy and paste a segmented control, you will now be able to open or compile the `.xib` or `.storyboard` file. (17071898)
- When adding an IBOutlet by Ctrl-dragging from an Interface Builder document to a Swift file, you can now set Storage to `strong` by adding the keyword. This overrides the outlet's implicitly weak behavior. Please note that outlets to AnyObject cannot be made strong in this release. (17047306)
- Interface Builder now supports declaring outlet collections in Swift classes. (15607242)
- It is now possible to uncomment indented line comments and comments mixed with blank lines. (9349394, 10311436)
- The Comment Selection/Uncomment Selection command in the Editor menu is now compatible with more languages. (8274382)
- Now you can toggle comments in a selection that includes the last line of the file. (9358280)
- The toolbar on the bottom of the Interface Builder canvas no longer changes to white text and icons on a white background.
- Images from asset catalogs in projects with a minimum deployment target of iOS 7 or OS X 10.9 will be available in apps running on iOS 7 and 8 and OS X 10.10 and 10.9. (17029658)

- NIBs containing an outline or table view with “source list” highlight style now compile. (17027302)

- `NSView` subclasses will now render as live views on the Interface Builder canvas even if they depend on setup in `-prepareForInterfaceBuilder` or `-layout`. (17081845)

- When finding text using a regular expression, it is now possible for a match to span multiple lines. The meta-character “.” still excludes newlines, but a character class may include every character, e.g. “[\D\d]”. (11393323)

GPU Tools

- The Metal Compiler build option “Produce Debugging information” now defaults to the correct value in new projects. Metal debugging information is required to view & profile pre-compiled Metal shaders.

Instruments

- Instruments will now find symbols when using Time Profiler with iOS Simulator. (16977140)

- The Automation instrument no longer requires enabling the "UI Automation" setting on the targeted iOS device. (16732303)

- Profiling Tests with Instruments now works for iOS Devices. (17057896)

- Profiling a Swift application with Allocations, Leaks or Zombies now works with 32-bit simulators. (17086159)

iOS Simulator

- When switching between resizable and regular devices in the iOS Simulator, the window contents no longer will become misaligned. (17023589)

- If a resizable device is being used on iOS Simulator, keyboard input will now go to the intended text field. (17024326)

- The resizable iPhone now works. (17022386)

- Logging into Game Center from the Settings application will no longer result in the error (16901415):

```
Unable to connect to server. The operations couldn't be completed. (Cocoa error 4097)
```

- You can now log into an iCloud account in the Simulator. (17135006)

- The “Toggle In-Call Status Bar” menu option in the Hardware menu now works. (17094855)

Testing

- Running tests on devices with iOS 7.1 installed now works. (17028705)

Xcode Server

- Xcode Server bots that run on both 32-bit and 64-bit simulators will fail.

  Workaround: configure bots to run on either only 32-bit simulators or only 64-bit simulators. (17329878)

- Xcode can now configure bots to authenticate with source control repositories using SSH keys. (16512381)

- Xcode 6 beta 3 connects to OS X Servers running Xcode 5-based Xcode services. (16745067)

- When setting up OS X Server and choosing which installed Xcode to use for building with Xcode Server, you will no longer be presented with a dialog requesting that you agree to the license agreement. (17031625)

- Simulators are now supported for use with Xcode Server. (17042416)

- When the Xcode Documentation window is open, the CPU usage will no longer be held at a constant high rate. (17074404)

- When the firewall is enabled, Xcode Server will now be reachable outside your network. (16544226)

.

Deprecation of OCUnit and SenTestingKit.framework

OCUnit and the SenTestingKit framework are deprecated and will be removed from a future release of Xcode. Source code using OCUnit will generate warnings while being compiled. Developers should migrate to XCTest by using the Edit > Refactor > Convert to XCTest command. For more information, see *Testing with Xcode* on developer.apple.com.