Xcode Release Notes

About Xcode 6 Beta

Supported Configurations

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

Xcode 6 includes SDKs for OS X versions 10.9 and 10.10, and iOS 8. 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-Beta4.app file to your Applications folder.

From within Xcode you can 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.

This Xcode beta does not support upgrade or migration of existing continuous integration services.

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

Unicode String improvements

The `String` type now 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 now 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)

Known issues in Xcode 6 beta 4

Swift

• You cannot conditionally assign to a property of an optional object. (16922562)

For example, this is not supported:

```
let window: NSWindow? = NSApplication.sharedApplication.mainWindow
window?.title = "Currently experiencing problems"
```

Similarly, you cannot modify the underlying value of a mutable optional value, either conditionally or within a force-unwrap:

```
tableView.sortDescriptors! += NSSortDescriptor(key: "creditName",
ascending: true)
```

Workaround: Test the optional value explicitly and then assign the result back:

```
if let window = NSApplication.sharedApplication.mainWindow {
   window.title = "Currently experiencing problems"
 }
tableView.sortDescriptors = tableView.sortDescriptors! +
NSSortDescriptor(key: "creditName", ascending: true)
```

- Properties of values typed as `AnyObject` may not be directly assigned to. The type of property access is an implicitly unchecked optional, so forcing the property access using '`!`' will result in the creation of an r-value, which cannot be assigned to. (15233922)

  Workaround: Cast the value of type `AnyObject` to the intended type, store it in a separate value, then assign it directly to the properties of that value. For example:
  ```
  var mc: MyClass = someAnyObject as MyClass
  mc.foo = "reassign"
  ```

- Value types, that is, structs and enums, that recursively reference themselves via a member type result in an assertion raised during code generation. (16423940)

- Swift does not support object initializers that fail by returning null. (16480364)

  Workaround: If there is a factory method, use it instead. Otherwise, capture the result in an optional. For example:
  ```
  let url: NSURL? = NSURL(string: "not a url")
  ```

- All equatable types can currently be compared with `nil` because the equatable type is implicitly promoted to an optional through value-to-optional conversion. This allows comparisons such as `someString != nil` to be accepted, even though `someString` can never be `nil`. (16848110)

- Optionals wrapping a value conforming to the LogicValue protocol (for example, `Bool?`) can be very confusing to work with. (17110911)

  Workaround: Avoid using optionals in a boolean context directly. Instead of:
  ```
  if value { ... }
  ```

  Use explicit comparisons to be precise about what should be tested, such as:
  ```
  if value != nil { ... }
  ```

  and:
  ```
  if value == true { ... }
  ```

- Nested functions that recursively reference themselves or other functions nested in the same outer function crash the compiler. (11266246)

  For example:
  ```
  func foo() {
     func bar() { bar() }

     func zim() { zang() }
     func zang() { zim() }
  }
  ```

  Workaround: Move recursive functions to the outer type or module context.

- A generic class can now define a stored property with a generic type, as long as the generic class is not made available to Objective-C (that is, is not marked as `@objc` or subclassed from an Objective-C class). (16737510)

```
class Box<T> {
    // "value" is a stored property whose type is the generic type
parameter T
    let value: T
    init(value: T) {
        self.value = value
    }
}
```

  Workaround: If you need to make such a class available to Objective-C, use an array for storage:

```
objc class ObjCBox<T> {
    let _value: T[]
    var value: T { return _value[0] }
}
```

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

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

Swift REPL

- To run the Swift REPL from the command line, pass the path to your beta Xcode to `xcode-select --switch`, then use `xcrun` to call the `swift` command line tool. (17159031)

```
$ sudo xcode-select -s <PATH/TO/Beta/Xcode6-Beta3.app>
$ xcrun swift
```

- Structs defined in the Swift REPL cannot be constructed using their implicit constructors. (17670241)

  Workaround: Provide an explicit constructor. For example:

```
struct foo {
  var a : Int
  var b : Int
  init(a: Int, b : Int) {
    self.a = a
    self.b = b
  }
}
```

Playgrounds

- Console output for OS X playgrounds may include an error: "Unable to create symlink at ...". This error can be safely ignored. (17669478)

- iOS playground execution is not sandboxed. (17437417)

- Only one Playground may be open at the same time. (17734742)

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.

- xibs with QCViews may not build correctly in this release. (17544013)

- 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.

- 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.

- If you set a Swift subclass of NSValueTransformer as a binding's value transformer, the XIB or storyboard will contain an invalid reference to the class, and the binding will 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.

Testing

- Unit tests written in Objective-C cannot currently import the Swift generated interfaces header ("$(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;'.

- Setting performance baselines from the test report may present an error if the user has not yet opened the Test navigator. (17039811)

  Workaround: Open the Test navigator, then set the performance baseline.

- Setting a bot is to run against "All iOS Simulators" will fail with a message: "Could not test because no simulators were available". (17559029)

- After installing Xcode Server and configuring bots, attempting to log in to the system at the login window may fail.

  Workaround: Log in immediately when the log in window appears. Alternatively, ssh into the server, kill the loginwindow process, then try logging again at the login window. (15081683)

- Xcode does not run individual tests on an iOS Device if the test target has spaces in its name. (16955715)

- Xcode does not run individual tests on an iOS Device if the test target has spaces in its name.

- When the Xcode service is disabled, bots will continue to integrate. (17633884)

- Devices and simulators may not appear in Xcode Server immediately.

  Workaround: In Terminal, execute, `sudo killall xcsdeviced`

App Extensions

- Debugging Keyboard Extensions is not supported. (16879317)

- OS X Action extensions created from the OS X Swift Action app extension template may not appear, and crash with a "Cannot form weak reference to instance..." log message. (17689219)

  Workaround: In Action extensions created from the template, remove the "weak" keyword from the myTextView outlet in ActionViewController.swift.

- 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.

- When 'Ask on Launch' is selected as the executable for app extension and 'Debug Executable' and 'Debug XPC Services and Extensions' are unchecked, you may see an error dialog and not be able to launch. (17542862)

  Workaround: Check the 'Debug XPC Services and Extensions' checkbox and try again.

- 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 e.g. "com.thirdparty.foo.keyboard (123)" via Instruments' target chooser under the running simulator device.

- Creating a Document Picker extension with the 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.

- After updating a previously installed Today extension, the extension may no longer launch. (17241004)

  Workaround: Reboot the iOS device.

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 now 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)

- Validating archives does not work. (16891311)

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

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

- Projects created from the OS X SceneKit project template may crash on launch. (17633598)

CloudKit

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

Compiler

- 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 now defined as __fp16. (15506420)

IOS Simulator

- Clicking in the space around an app in the resizable iPhone sim will crash the app. (17022609)

  Workaround: Only click within the app's window.

- In the Simulator, Darwin notifications sometimes stop getting delivered to apps that have been suspended and resumed. (16823204)

Xcode Server

- Bots configured to create archives for Mac projects will silently fail to create archives. (17700710)

Important Changes, Issues Resolved in Xcode 6 Beta 1 - 3

Swift Language

- The online *Swift Programming Language* documentation and book has 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 no 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.