Xcode Release Notes

About Xcode 6 Beta 3

Supported Configurations

Xcode 6 beta 3 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-Beta3.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.

Important Changes, Issues Resolved in Xcode 6 Beta 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)

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

- 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.

- The Metal GPU Frame Debugger will now capture a frame when the app attempts to reuse a `CAMetalDrawable` from a previous frame. (16854598)

Instruments

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

- Start/Clear/End range buttons in the toolbar for Instruments' workflow for setting, modifying, and clearing track view inspection ranges. (16513433, 11956549)

- 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)

Known Issues in Xcode 6 Beta 3

Swift Language

- Swift currently lacks an analog for Objective-C's `#pragma mark`. (14768427)

- Generic classes with non-fixed layout are now supported for classes without Objective-C base classes or the `objc` attribute. Classes have non-fixed layout when they contain a stored property of a generic type variable type, as in:

```
class Box<T> {
   // Stored property of type T involves non-fixed layout
   let value: T
 }
```

  or if they contain structs, tuples, or enums that have stored properties or cases involving type variable types. This form of layout is still not implemented for objc classes; as a workaround, an array can be used for storage: (17139631 & 16737510)

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

- Access control (public/private members) is not enabled in this seed. (15747445)

- Properties of values typed as `AnyObject` may not be directly assigned to.

  Workaround: Cast the value of type `AnyObject` to the intended type, and store it off to a separate value. You will then be able to assign directly to properties of that value. (15233922)

  For example:

```
var mc: MyClass = someAnyObject as MyClass
mc.foo = "reassign"
```

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

  Workaround: If there is a factory method, use that instead. Otherwise, as a workaround, immediately capture the result in an optional. (16480364) For example:

```
let url: NSURL? = NSURL(string: "not a url")
```

- It is not possible to build static libraries which contain Swift code in this release. (17181019)

Playgrounds

- Playgrounds are not currently sandboxed. They run with the full permissions of the logged in user. Care should be taken before opening playgrounds obtained from others. Before opening the playground you can inspect the code in a playground by looking in ".swift" files inside the

playground wrapper. Control click the playground in Finder and choose Show Package Contents to see the contents of that playground. (16773467)

- Certain usages of `UnsafePointer<T>` in playground code may cause results to not appear at or beyond the line involving such usage.

  Workaround: Use `UnsafePointer<T>` only in subexpressions. (17055083)

Swift REPL

- To run the Swift REPL you must first make the Xcode6-Beta3.app app bundle the default for command line tools on your host. In a Terminal window run the command:

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

  For example:

  ```
  > sudo xcode-select -s /Applications/Xcode6-Beta3.app
  ```

  Thereafter to launch the Swift REPL, in the Terminal window, type:

  ```
  > xcrun swift
  ```

General

- Attempting to open a SpriteKit Scene SKS file in the Version Editor may crash Xcode. (17046728)

- In the OS X doc set, the Uniform Type Identifiers Reference includes zero-width spaces (U +200B) in many of the listed system UTIs. These and other hidden Unicode characters are not shown visibly in Xcode. Do not copy and paste these UTIs directly into Xcode. (15884891)

- The container group ID set by a document picker template does not conform to the requirements as documented out in the *Entitlement Key Reference*. The prefix consisting of the development team ID is missing. This impacts the values of the `com.apple.security.application-groups` entitlement and of the `NSExtensionFileProviderDocumentGroup` Info.plist key in the file provider extension. (16869846)

- Validating archives does not work. (16891311)

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

- When opening an image inside an Xcode workspace window, the preview may be blank.

  Workaround: Either open the image in a separate window by double-clicking the file in the Project navigator or go to View > Hide Toolbar. (16876910)

Building

- Xcode will no longer pass options in the build setting `OTHER_LDFLAGS` to libtool when building static libraries, nor will it pass options in `OTHER_LIBTOOLFLAGS` to the Mach-O linker when building any other kind of product. Previously all options in both settings would be passed to both tools. Make sure that options are in the correct build setting for the product type, static library or other, being built. (4285249)

- App Extension targets and XPC Service targets no longer have their product name prefixed with the bundle identifier prefix. (16941769)

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

  Be aware of this limitation if you intend to hand off a command-line tool you have built to someone else. (16866827)

- It is not possible to build static libraries which contain Swift code in this release. (17181019)

Command-Line Tools

- On-demand command-line tools are not available for OS X version 10.10 beta.

  Workaround: Manually download the OS X 10.10 command line tools from developer.apple.com. (17034669, 17074662)

Compiler

- An ABI incompatibility exists for ARM and ARM64 code that uses the `float16_t` type defined in the `arm_neon.h` header. In previous versions of the compiler, `float16_t` was defined as `uint16_t; float16_t` is now defined as `__fp16`. Most code should still be compatible with that change. There could be problems when trying to link C++ code compiled with an older compiler, where a `float16_t` function argument will be handled differently in the mangled function name. (15506420)

- The `libc++` headers in Xcode 6 include a change to make `std::pair` have a trivial constructor. This fix is important for performance and compliance with the C++ standard, but it changes the ABI for C++ code using `std::pair`. This issue does not affect the `libc++` library installed on OS X and iOS systems, because that library does not expose any uses of `std::pair` in its API; this is only a concern for your own code and any third-party libraries that you link with. As long as you rebuild all of your code with the same version of Xcode, there will be no problem.

  Workaround: If you need to maintain binary compatibility with a previous version of `libc++`, you can opt into keeping the old ABI by building with the `_LIBCPP_TRIVIAL_PAIR_COPY_CTOR` macro defined to zero; for example, add `-D_LIBCPP_TRIVIAL_PAIR_COPY_CTOR=0` to the compiler options. (15474572)

Debugger

- Debugging Swift code in 32-bit apps on iOS 7 will fail and can crash Xcode. This problem affects both devices and simulators. (17553837)
- When updating a previously installed Today extension, the extension may not launch anymore.

  Workaround: Reboot the device. (17241004)

- iOS application extension debugging in the simulator on OS X 10.10 will sometimes crash Xcode.

  Workaround: Rebuild the project from clean. (17527929)

- Debugging Share or Action app extensions does not work in this release. You will see an error message similar to: "The app <app_name> on <device_name> quit unexpectedly. Message from debugger: Terminated due to signal 9" (17303593)

- Location simulation is not functional running on OS X 10.10. (16586648)

- When your app is paused and you select a stack frame without debug symbols, Xcode may not show you the correct disassembly.

  Workaround: Type "disassembly" in the Console Area, like this: `(lldb)disassembly` (17044846)

- When the scheme "Ask on Launch" option is selected as the executable for app extension hosting and "Debug executable" and "Debug extensions and XPC services" are unchecked, you may see an error dialog and not able to launch.

  Workaround: Check the "Debug XPC Services and Extensions" checkbox and try again. (17023999)

- Debugging an Action extension on iOS Simulator via Xcode's Build & Run method does not work. (17029783)

  Workaround: Debug the action extension on an actual iOS device.

Editing User Interfaces

- In Swift files, properties with the `@IBInspectable` attribute and Swift-only types such as `String` are not shown in the Attributes inspector in Interface Builder.

  Workaround: Change properties' types from Int to NSInteger, from Float to CGFloat, and from String to NSString. For other types, use the User-Defined Runtime Attributes section of the Identity inspector. (17558064)

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

- 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.

  Workaround: Either enter a mangled class name into the Value Transformer field, or add the `@objc(…)` attribute to the NSValueTransformer subclass. (17495784)

- Live views with the same name in two different projects opened simultaneously will not render correctly.

  Workaround: Only open one project at a time if it contains a designable class with the same name as another project. (15941330)

- When finding text 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", which was introduced in Xcode 4.4. 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)

- 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.

  Workaround: Select each reference, then in the Custom Class inspector, clear the Class field, save, and reenter the class name. (17153630)

- Forming a binding connection between a control in an OS X Storyboard scene and the user defaults controller will crash when saving the document.

  Workaround: Add a User Defaults controller to the dock of the scene and bind to it instead. (17043536)

- When running a Storyboard based project on OS X, failed outlet connection warnings are output to the Console. These warnings may be ignored. (17027315)

- Unwind segue actions declared in Swift classes are not recognized by Interface Builder.

  Workaround:

  1. Change `class MyViewController` to `@objc(MyViewController) class MyViewController`

  2. Add an Objective-C header with a category on `MyViewController` that redeclares the segue action. It should look something like this:

```
@interface MyViewController (Workaround)
(IBAction)unwindToMyViewController:(UIStoryboardSegue *)segue;
  // or whatever the unwind action is called
@end
```

  3. In the storyboard, select the instance of `MyViewController`, clear its custom class, then set it back to `MyViewController`. You should now be able to connect to the action by Control-dragging to a view controller's Exit icon. (15966387)

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

  Workaround: Declare the outlet's type as `AnyObject` or `NSObject`. (17023935)

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

  Workaround: Pick a `.xib` file that includes a `NSCollectionView` and load it into a storyboard based view. (17009377)

GPU Tools

- Display of the frame buffer on the device while using the GPU Frame Debugger to debug Metal applications is disabled in this release. (16936055)

Instruments

- Profiling Unit or Performance Tests in Instruments only works reliably when Instruments isn't already running.

  Workaround: Quit Instruments just before each "Profile Test" workflow. (17033594, 17034363)

- Using xpc_service_set_attach_handler to profile App Extensions for keyboard isn't working.

  Workaround: Trigger the Keyboard extension to load and then attach to the running instance—for example, `com.thirdparty.foo.keyboard (123)`—via Instruments' target chooser under the running simulator device. (16946100)

- Profiling Unit or Performance Tests in Instruments only works reliably when Instruments isn't already running. (17034363)

  Workaround: Quit Instruments just before each "Profile Test" workflow.

- Profile Test with Instruments can fail due to a linker error.

  Workaround: Change the build setting for your App or Framework to

  `GCC_SYMBOLS_PRIVATE_EXTERN=NO`

iOS Simulator

- If, after running Xcode, you rename Xcode.app the Simulator stops working.

  Workaround: Reboot or change revert the name change. (16646772)

- Changing keyboards in Settings > General > Keyboard may require you to relaunch your app before the new settings are observed. (16891121)

- The resizable simulator will not resize while social, photo sharing, or action extension items are displayed.

  Workaround: Dismiss the view, app switch to another app and back. (17256834)

Testing

- When using `xcodebuild` to run tests, the first test run for XCTest targets executing on iOS Simulator will fail. Subsequent test runs succeed. (17007178)

- 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. Tests for Swift code should be written in Swift.

  Workaround: Tests written in Objective-C for framework targets can access the Swift generated interfaces by simply importing the framework module using `@import FrameworkName;`. (16931027)

- Xcode will not run individual tests on a device if the test target has space in its name. (17055187 & 16955715)

- Setting performance baselines from the Test Report may present an error if the user has not yet opened the Test Navigator.

  Workaround: Open the Test Navigator and try again. (17039811)

Xcode Server

- Developer mode is not enabled by default. To enable Developer mode, run the following command from Terminal and then restart the computer: (17454648)

  ```
  sudo /usr/sbin/DevToolsSecurity -enable
  ```

- Importing developer accounts via the Accounts preferences crashes Xcode.

  Workaround: Use Xcode 5.1.1 to import and export developer accounts. (17509786)

- Xcode bots that build for iOS will fail to archive due to code signing issues. (17485516)

- Some builds may not complete as expected.

  Workaround: Rerun the integration. (17512198 & 17368481)

- When a bot is set up to run against "All iOS Simulators", the build fails 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 host machine and kill the loginwindow process. Then try logging again at the login window. (15081683)

- Bots configured to test on multiple devices but that are not configured to build from clean may see this error:

  ```
  Linker command failed with exit code 1: ld: can't link with a main executable file.
  ```

  Workaround: Edit the bot and configure the cleaning setting to be `Always`, or test on a single device only. (17040693)

- Xcode Server tests of Mac apps may fail intermittently with an error indicating headless testing is unavailable.

  Workaround: Restart the machine.

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.