# Xcode Release Notes

## About Xcode 6.1

### Supported Configurations

Xcode 6.1 requires a Mac running OS X 10.9.4 or OS X 10.10.

Xcode 6.1 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 Xcode-Beta.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 release, you need either OS X 10.9.4 with OS X Server 3.2 or OS X 10.10 with OS X Server 4.0.

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

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.

## 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.

# New in Xcode 6.1 Beta

## Swift Language

• A large number of Foundation, UIKit, CoreData, SceneKit, SpriteKit, Metal APIs have been audited for optional conformance, removing a significant number of implicitly unwrapped optionals from their interfaces. This clarifies the nullability of their properties and arguments / return values of their methods. This is an ongoing effort.

These changes replace `T!` with either `T?` or `T` depending on whether the value can be null or not respectively. If you find a case that was changed incorrectly, please file a radar and include the tag "#IUO" in the subject line.

If you encounter a method for which the return value is incorrectly considered non-nullable, or a property that is incorrectly considered non-nullable, you can work around the problem by immediately wrapping the result in an optional:

```
var fooOpt: NSFoo? = object.reallyMightReturnNil()
if let foo = fooOpt { … }
```

Please be sure to file a bug about these cases. Please do not file feature requests about APIs that are still marked as `T!`. We know about them.

• Initializers can now fail by returning `nil`. A failable initializer is declared with `init?` to return an explicit optional or `init` to return an implicitly-unwrapped optional. For example, you could implement `String.toInt` as a failable initializer of `Int` like this:

```
extension Int {
  init?(fromString: String) {
    if let i = fromString.toInt() {
      // Initialize
      self = i
    } else {
      // Discard self and return 'nil'.
      return nil
    }
  }
}
```

The result of constructing a value using a failable initializer then becomes optional:

```
if let twentytwo = Int(fromString: "22") {
  println("the number is \(twentytwo)")
} else {
  println("not a number")
}
```

In the current implementation, `struct` and `enum` initializers can return `nil` at any point inside the initializer, but class initializers can only return `nil` after all of the stored properties of the object have been initialized and `self.init` or `super.init` has been called. If `self.init` or `super.init` is used to delegate to a failable initializer, then the `nil` return is implicitly propagated through the current initializer if the called initializer fails. (16480364)

- Integer types now provide a `truncatingBitPattern` initializer for performing an integer truncation from another integer type.

- Custom operators can now contain the `?` character.

### General

- The HomeKit Accessory Simulator is now available as part of a separate downloadable package called Hardware IO Tools for Xcode, available at http://developer.apple.com. (17738621)

- The Printer Simulator is now available as part of a separate downloadable package called Hardware IO Tools for Xcode, available at http://developer.apple.com. (17014426)

- 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)

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

- 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)

- Xcode 6 introduced a new feature enabling the use of XIBs and Storyboards as launch screens. Applications can take advantage of this feature to provide a single launch screen that adapts to different screen sizes using constraints and size classes. These launch screens apply to iOS 8.0 and later. Applications targeting release prior to iOS 8.0 will need to also supply traditional launch PNGs via an asset catalog. Without launch PNGs for iOS 7 and earlier, applications will run in the retina 3.5" compatibility mode on retina 4" displays. (18000959)

### Build System

- 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)

- 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)

• Bundles (including frameworks, applications, and other bundles) which are added to a Copy Files build phase to embed the bundle in the current target's product will have their Headers and PrivateHeaders directories removed when they are copied. This is done to help ensure that header content in embedded bundles is not accidentally shipped to end users. This does not affect such items which are already in Copy Files build phases in existing projects. This will affect any file or directory inside the item being copied which is named exactly 'Headers' or 'PrivateHeaders'. To have a target opt out of this behavior, set the build setting REMOVE_HEADERS_FROM_EMBEDDED_BUNDLES to YES. To have an existing target opt in to this behavior, remove and then re-add the desired file references to the Copy Files build phase in question.

### Xcode Server

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

• Xcode Server prunes older integration data based on available disk space. (16535845)

## Issues Resolved in Xcode 6.1 Beta

### Swift Language

• The relational operator == now works on enum values even if the enum is declared in another file. (18073705)

• Using a Swift Dictionary with `Int64` or `UInt64` types as keys will no longer cause traps on 32-bit platforms when attempting to insert a key whose value is not representable by an `Int32`. (18113807)

# Known Issues in Xcode 6.1 Beta

## Swift Language

• Properties of values typed as `AnyObject` may not be directly 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"
```

• Private entities with the same name and same type will conflict even if defined in different files within the same module. (17632175)

• Nested functions that recursively reference themselves or other functions nested in the same outer function will 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, it is not marked as `@objc` or subclassed from an Objective-C class. (16737510) For example:

```
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] }
}
```

• The build step which embeds the Swift standard libraries in a bundle only runs for application product types, and only if the application itself, independent of any embedded content, contains Swift source files. When building an application that itself does not contain Swift source files, but embeds other content like frameworks, XPC services, app extensions, etc. that do contain Swift code, you must set the build setting:

Embedded Content Contains Swift Code" (EMBEDDED_CONTENT_CONTAINS_SWIFT)

That way the Swift libraries will be included in the application. (17757566)

• A mixed Swift and Objective-C project may not rebuild fully after a header is changed. (17963128)

Workaround: Do a clean build.

• 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 will need to be made by hand.

## Playgrounds

• iOS Playgrounds do not support displaying animated views with the XCPShowView() XCPlayground API. (17848651)

• A weakly-linked symbol in a Playground may cause a compilation error. (18000684)

• In Playgrounds, `println()` ignores the Printable conformance of user-defined types. (16562388)

## Testing

• Dead code stripping may remove public declarations from Swift application targets which are needed by unit testing. (18173029)

Workaround: Turn off dead code stripping in configurations where you are building unit tests.

## General

• Docsets sometimes do not show up in Xcode's documentation viewer after downloading. (18260884)

Workaround: Quit and relaunch Xcode.

• 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, but float16_t` is now defined as `__fp16`. (15506420)

### Build System

• Targets with a minimum deployment target of iOS 8.0 that use Cloud Documents must manually specify the com.apple.developer.ubiquity-containers entitlement in their entitlements file with a clone of the values for the corresponding CloudKit services entitlement. (18247931)

### Testing

• Unit tests written in Objective-C cannot 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;`.

### Asset Catalogs

• If your project's only asset in as asset catalog, the build will fail with the error '.../Contents/Resources" does not exist.' (17848595)

Workaround: Add at least one non-asset catalog resource to your project.

### Interface Builder

• 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.

• 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, clear the Class field in the Custom Class inspector, save, and reenter the class name.

• Interface Builder will only show custom geometry overrides (e.g. -intrinsicContentSize) in a designable subclasses in iOS documents. (17024838)

• If a storyboard or XIB uses size classes and base localization and the build target is Universal and targets iOS 7.0, the storyboard or XIB will not localize correctly. (18087788)

Workaround: Add a ~iphone and ~ipad variant of each strings file used by the affected storyboards and XIBs. You can automate this step with a build phase. Select the application target in the project editor, then go to the Build Phases tab. Add a new Run Script phase with the following contents:

```
# Go to the app bundle.
cd "${BUILT_PRODUCTS_DIR}/${UNLOCALIZED_RESOURCES_FOLDER_PATH}"

for f in "$PWD"/*.lproj/*.strings; do
    # If the .strings file name doesn't already specify a device...
    name=$(basename "$f" .strings)
    if [[ "${name%%~iphone}" == "$name" && "${name%%~ipad}" == "$name" ]]; then
        # If there is a corresponding .nib in Base.lproj...
        if [[ -e "Base.lproj/$name~iphone.nib" ]]; then
            # Symlink device-qualified file name to the unqualified file.
            ln -sf "$f" "${f/%.strings/~iphone.strings}"
        fi
        # Do likewise for iPad.
        if [[ -e "Base.lproj/$name~ipad.nib" ]]; then
            ln -sf "$f" "${f/%.strings/~ipad.strings}"
        fi
    fi
done
```

## Debugging

• Data structures containing bitfields may display incorrectly in the debugger when referenced from Swift code. (17967427)

• View Memory for Swift data structures in the debugger may show memory location zero. (17818703)

Workaround: Pass the structure to a function expecting a reference to an `UnsafePointer<Void>`, and print it within the function. Enter this address as the memory location to view.

• Expressions like `expr`, `p`, and `print` that are evaluated from the LLDB prompt in the debugger console will fail on 32-bit iOS devices. However, they will work on 64-bit devices and the iOS simulator. (18249931)

## iOS Simulator

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

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

• Testing on the iOS simulator may produce an error indicating that the application could not be installed or launched. (17733855)

Workaround: Re-run testing or start another integration.

### Source Control

• Opening the Source Control menu in Xcode when a project references a working copy that is not checked out sometimes causes crashes. (17905354)

Workaround: Delete the .xccheckout file within the project.

### Xcode Server

• After upgrading to Xcode 6, users will need to rejoin their ADC development team in OS X Server. (17789478)

### App Extensions

• 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.

• Creating an iOS 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.