Xcode Release Notes

About Xcode 7 beta

Supported Configurations

Xcode 7 beta requires a Mac running OS X 10.10.

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

Installation

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

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

Installing Xcode on OS X Server

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

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

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

Technical Support and Learning Resources

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

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

• http://developer.apple.com/xcode: The Xcode home page on the Apple Developer website provides information 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.

Deprecations and Removal Notices

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

Xcode 7 beta — New Feature Highlights

Platform Support

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

Swift Language

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

Objective-C Updates

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

Playgrounds

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

App Thinning

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

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

- **Slicing.** Artwork incorporated into the Asset Catalog and tagged for platform needs allows the App Store to deliver only what is needed for installation.
- **On Demand Resources.** Host additional content for your app on the iTunes App Store repository, allowing the app to fetch resources as needed using asynchronous download and installation.

Debugging

- **Energy Gauge for iOS.** Track energy usage with iOS 9 on a per-process basis.
- **Address Sanitizer.** Build instrumented Objective-C and C code to trap the sources of memory corruption problems.

Testing

- **User interface testing and recording.** Test applications at the user interface surface with elements, queries, and simulated events. The UI recording feature enables capture of UI actions into source to facilitate creating tests.
- **Code coverage.** Provides report information to evaluate a test suite for completeness.

Free Provisioning

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

Crash Logs

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

New Features in Xcode 7 beta IDE

Swift in Xcode

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

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

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

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

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

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

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

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

- Use the "disable safety checks" build setting to disable runtime checks.

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

Interface Builder

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Compiler

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

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

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

Build System

- The Xcode build system supports stale file removal of some types of build artifacts which were produced in a previous build, but have since been removed from the project. (19479602)

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

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

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

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

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

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

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

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

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

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

App Thinning

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

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

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

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

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

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

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

Debugger

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

- Address Sanitizer. Objective-C and C/C++ code is particularly susceptible to memory corruption issues such as stack and heap buffer overruns, use-after-free errors, and so forth. They can result in crashes or odd program behavior and could also lead to security exploits. Memory corruption bugs are difficult to track down because they are often hard to reproduce and the root cause can be far from the manifestation of the problem. When you enable the Address sanitizer in your Scheme Editor, Xcode 7 builds your app with added instrumentation so that these issues can be caught immediately, when they happen. Address sanitizer allows you to inspect the problem right at the place where the memory violation occurs; in many cases, it also provides other information needed to diagnose the bug such as relation of the faulty address to a valid object on the heap and its allocation/deallocation information. Address sanitizer has a small performance overhead; it is fast enough to be used with interactive applications. Address sanitizer is supported on OS X, in the iOS Simulator, and on iOS device. (10514936)

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

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

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

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

- While debugging your view hierarchy, Xcode allows you to focus on a portion of your app by double clicking on a view. To exit, click the (x) in the debug navigator or double click in the white space of the editor. (18553133)
- Data formatting for Swift types examined using “po” can be customized by implementing one or more new protocols: CustomStringConvertible, CustomDebugStringConvertible, CustomPlaygroundQuickLookable, and CustomReflectable. (18681736)
- Clang modules imported with “expr @import <modulename>” include access to macros and always-inlined functions like NSMakeRect(). (19575787)
- Swift expressions that throw errors in LLDB expression evaluation and Swift REPL sessions produce variables prefixed with $E that can be referenced in subsequent expressions. The expression evaluator also allows a relaxed syntax that does not require try before an expression that can throw. (20864812)
- Vector types in C and Swift are automatically formatted. (20294199)
- Address sanitizer integration is supported through the new “memory” command in LLDB. (21245519)
- Information about inheritance and members of types is available from the LLDB prompt with the “type lookup” command. (17307067)
- The energy gauge is available to iOS apps running on a device. (15032678)

Instruments

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

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

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

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

Testing

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

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

Find Navigator

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

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

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

Schemes

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

Resolved Issues in Xcode 7 beta — IDE

Interface Builder

- Interface Builder allows setting an identifier for an auto layout constraint, to assist in debugging unsatisfiable constraints. (13645911)
- Holding the Command (⌘) key while dragging to move a view will keep the dragged view in the same container, rather than making it a subview of a view it is dragged over. (18683271)
- Additional types of constraints can be created in Interface Builder including proportional width and height constraints to any superview such as the content view of a window or view of a view controller. (17011782)

Build System

- Fixed an issue which could cause Xcode to occasionally fail to rebuild some source files when their imported headers changed. (13697791)
- Improved the heuristic used to throttle the build system's parallelism under memory pressure. (20245603)
- Fixed an issue which could cause Xcode to use less parallelism when building than it should. (17665160)
- Xcode performance when reloading .xcconfig files from disk is significantly improved. (14136335)
- Fixed an issue which caused stopping a build to sometimes take much longer than it should. (10233505)
- A "Copy Files" build phase on a framework target with a destination of "PlugIns" will now copy its contents to the "PlugIns" directory in the framework rather than the "Resources" directory. (11488493)
- "Other Code Signing Flags" (OTHER_CODE_SIGN_FLAGS) build setting properly handles spaces. (18977465)

Instruments

- With iOS 9.0 devices, Leaks recordings should no longer get stuck "Analyzing Process." (18215045)
- When profiling Swift Apps, standard library symbols should be correctly symbolicated. (21131543)

General

- Run Script scheme actions take the run destination into account when passing environment variables, such as SDKROOT, that depend on the run destination. (19652312)
- When opening a workspace with more than 80 targets for the first time, Xcode prompt you to ask if you want Xcode to autocreate schemes for those targets instead of doing so automatically. (20877091)

Known Issues in Xcode 7 beta — IDE

Projects

- If you have accepted Xcode's "Upgrade to recommended settings" project modernization to adopt the new Product Bundle Identifier (PRODUCT_BUNDLE_IDENTIFIER) build setting, the Bundle Identifier field in the General tab of the target editor will no longer be editable. (20888003)

    Workaround: You can edit your bundle identifier in the Build Settings tab with the Product Bundle Identifier build setting.

- Sometimes projects may fail to build because a storyboard fails to compile with the error “A file with the name “IB” already exists”. (20771842)

Playgrounds

- Opening a Playground that was previously displaying the Version Editor may present an alert stating, “The file “MyPlayground.playground” couldn’t be opened because you don’t have permission to view it.” (20623808).

    Workaround: Dismiss the alert dialog and show the Standard Editor (Command-Return).

- Playgrounds sometimes show a window with no editor if they are opened as part of Xcode launching (20694143)

    Workaround: Close and re-open the playground.

- Executing an iOS Playground for the first time may take longer than subsequent executions. (21163503)
- For old-style playgrounds, the menu item Edit->Convert->To Latest Swift Syntax… is greyed out. (20902099)

    Workaround: Upgrade the playground before attempting to convert to Swift 2.0 by choosing Editor->Upgrade Playground…

- Playground pages may recompile each time they are navigated to, even when there are no changes. (21177662)
- Playgrounds live views may not show images when using the timeline slider. (21261310)

Source Editor

- Xcode may crash when editing source code while also viewing the generated interface in the assistant editor. (21211565)

    Workaround: Don’t edit sources while also viewing the generated interface.

Compiler

- dsymutil may print warnings about unresolved external types when compiling C++ projects with precompiled headers. (21170404)

  Workaround: Avoid using precompiled headers in cases where you see this problem.

Build System

- If you change the value of the PRODUCT_BUNDLE_IDENTIFIER build setting and then rebuild, the Info.plist will not regenerate. (21066965)

  Workaround: Perform a clean after changing the PRODUCT_BUNDLE_IDENTIFIER build setting.

- Deployment of 64-bit-only iOS apps to devices running iOS versions earlier than 8.0 does not work; it will fail with an error that the app "specifies device capability requirements which are not met". (21195657)

  Workaround: Configure your project to build for more than one architecture, as the problem only occurs when building *only* for arm64: Make sure that your target(s) have their Architectures build setting set to build for multiple architectures, and turn off Build Active Architecture Only for the target+configuration in question.

Simulator

- Xcode 7.0 beta does not support iOS 8.4 and earlier simulator runtimes. (20699475)

- When launching an app for the first time in the Simulator, the launch may hang. (20940542)

  Workaround: Stop and re-run the app.

- Background upload/download using NSURLSession will complete, but notifications will not badge the app to show completion when the app is in the background. (16532261)

  Workaround: Test on device to confirm works as expected.

- Printing from Simulator does not work with the iOS 9.0 beta runtime. (19754468)

  Workaround: Test printing from a device with iOS 9.0 beta.

- When launching an app in the sim, the launch screen will sometimes be in the wrong orientation. (20876494)

- On first launch of the Simulator, a ServerFileProvider crash will appear in the User/Library/Logs/DiagnosticReports folder. (20985383)

  Workaround: This crash is benign and should not effect production of any kind.

- iPad Air Simulator should have limited multi-tasking capabilities (overlay and PiP only). There is no difference in the iPad Air and iPad Air 2 simulators. (21051850)

- Playing video to a simulated external display does not function correctly with iPad Simulators. (17979778)

Workaround: Use an iPhone Simulator.

- Video playback is often incorrectly cropped or translated when playing back on a TVOut display in Simulator. (19394195)

    Workaround: Test video playback on a physical device

- Playing back video to a 1080p external display in Simulator will actually render at a lower resolution (18296724)

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

- If an External Display is open when you switch simulators, the sim will get stuck at the Apple Logo. (21224352)

    Workaround: Disable external display and reboot the simulated device.

- Restoring from TimeMachine backup after running Xcode 7 will result in no simulator devices listed in Xcode by default. (21215747)

    Workaround: Either create your desired devices from Xcode's Devices Window or delete ~/Library/Developer/CoreSimulator prior to running Xcode after a reboot.

- Simulator target are missing on first launch, after a clean install.

    Workaround: Quit Xcode and relaunch.

- Downloads for old Simulators are not available. (21060833)

Debugger

- When debugging your view hierarchy, the inspector will not update when selecting views in the debug navigator. (20748069)

    Workaround: Click on the view in the editor area.

Address Sanitizer

- There are known problems with using the address sanitizer on app extensions on iOS devices (21118074).

GPU Debugger

- Shader profiling is disabled when capturing a frame from Metal applications using Nvidia GPUs. (21165399)

Instruments

- Launching Instruments without first launching Xcode.app and installing required components suggested by Xcode may result in Instruments.app crashing on launch. (20844043)

Testing

- Xcode does not show code coverage information for source files in static libraries. (15605406)

  Workaround: Add the source files directly to application or framework targets.

- Code coverage does not work with UI testing. (20966994)

- Long assertions in the test report are truncated. (21083089)

  Workaround: The full text of assertions can be found in the Logs tab.

- The side bar of a source editor for a test class will not always reliably show or update test indicators when the Test navigator is not showing. (20553093)

  Workaround: Revealing the Test navigator should resolve this issue.

- UI testing cannot identify elements using information from their accessibility title element. (20409319)

  Workaround: Set an accessibility identifier instead.

- UI testing may fail to find elements with names containing decomposable Unicode characters. (20804391)

  Workaround: Use NSPredicate to explicitly match against key paths including precomposedStringWithCanonicalMapping, e.g. title.precomposedStringWithCanonicalMapping.

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

- Recording UI tests do not work if the device or simulator is in landscape orientation. (21096781)

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

- Auto-correct interferes with UI tests that type text, making results unpredictable. (21106884)

  Workaround: Disable auto-correct on devices and simulators used for UI testing.

- The “Record UI Test” button sometimes causes Xcode to hang if the applications is not already running. (21224350)

  Workaround: Launching the application before clicking the “Record” button should resolve this.

- The “Edit ▸ Convert ▸ To XCTest…” command will fail with an error that SenTestingKit.framework cannot be found. (21251738)

  Workaround: Use an older Xcode to convert tests written using OCUnit to XCTest.

- The OCUnit to XCTest migrator leaves test targets still linking against SenTestingKit.

  Workaround: Manually remove the SenTestingKit linkage.

- After running performance tests within Xcode, the ability to inspect results and edit target criteria is missing. (21252208)

  Workaround This functionality is available in the Source Editor. After running the performance tests, open the test class in the Source Editor and click on the Performance Result near the right side of the editor. This should reveal the Performance Result and allow editing of the Baseline and Max STDDEV values.

- UI recording's generated code for Swift is incorrect for multiple modifiers. (21183420)

  Workaround: Re-write option sets to be surrounded with [] brackets instead of parentheses and delimited using commas ','

- API and data needed to automate complex controls such as sliders and date pickers is not available. (20577276)

- The Automation Instrument may hang when starting capture in the Simulator. (21196158)

  Workaround: Quit the Simulator and re-try.

Interface Builder

- Distance guides, which are displayed when holding the Option key while hovering over a view hierarchy in Interface Builder, may not appear when using Xcode 7.0 on Mac OS X 10.11. (20758551)

- The Interface Builder Connections Inspector does not show a view outlet for Swift ViewController subclasses if they are marked with @objc but do not have an exported class name(20909753)

  Workaround: Add the exported class name in parens, such as @objc(MyClass)

- Xcode crashes when searching in a project that has an OS X Storyboard with a Custom Segue. (21154156)

  Workaround: Use a Find scope to filter out Storyboards or remove the Custom Segue.

- Items in Xcode's Find Navigator scope picker outline view are invisible on 10.11 (20275994)

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

- Storyboard References do not work with WatchKit storyboards. (21191010)

On Demand Resources

- Adding or removing tags for On Demand Resources content may not take effect. (21169688)

  Workaround: Delete the app from the device or simulator and then re-install the app.
- Xcode may emit a warning that "Application Slicing is disabled for this seed of Xcode if size classes are used." (21278829)

  Workaround: Set ENABLE_ONLY_ACTIVE_RESOURCES to disable the warning.

Localization

- During XLIFF export or import, NSLocalizedString macro issues or empty strings files may result in an error "The data couldn’t be read because it isn’t in the correct format”. (21101899)

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

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

- Localization stringsdict files may result in an error "failed to read strings file" during XLIFF export or import. (21113196)

  Workaround: Remove the stringsdict files from your target's copy files phase.

Source Control

- When committing a removed file to a Git repository, the file will be deleted on disk but not removed from the Git repository. (21222457)

  Workaround: Commit deleting files from a released version of Xcode or from Terminal. The files will remain missing from the local repository after the commit in Xcode 7 and can be committed from another Xcode or Terminal.

- Existing Source Control accounts may appear with empty passwords after launching.

  Workaround: Re-enter the password.

Xcode Server

- Continuous integration bot email notifications may not send by default on some networks. (21189162)

  Workaround: Configure a custom SMTP relay server using `sudo xcrun xcscontrol --configure-email-transport server`.

- Existing Xcode Server instances upgraded to Xcode 7 may take some time to migrate and update bot history data. (21252055)

Core Data

- The mapping model template assistant is not available. (20267140)

  Workaround: Use an earlier version of Xcode to create a mapping model.

- Generating NSManagedObject subclasses for multiple entities crashes Xcode. (20638023)

  Workaround: Repeat operation until it works or generate a subclass for each entity individually.

Free Provisioning

- Accounts with expired Apple Developer Program memberships may not be able to use free provisioning. (20486606)

  Workaround: Create a new Apple ID and enter it into Xcode's Accounts preference pane.

- The list of teams displayed within Xcode's Accounts preference pane may not update to show your new free provisioning team. (19775448)

  Workaround: Remove your Apple ID and add it again in Xcode's Accounts preference pane.

- When logging in via the Preferences window's Accounts pane, any standard Apple ID will allow you to log in and use Free Provisioning. Such logins will not be subject to the Apple Developer Program License Agreement. (21243707)

- You may be asked to select between a free membership and your company team membership when configuring apps for distribution. (20276021)

Devices

- An error message incorrectly states "To fix this issue, you need to add an Apple ID account that is enrolled in the Apple Developer Program." (21245164)

  Workaround: Users can fix problems without an Apple ID account that is enrolled in the Apple Developer Program.

watchOS

- There are known performance issues developing watchOS 2 applications on devices. Significant delays during Watch app installation and launching with the debugger are also known and will be addressed in future seeds.

- You may experience installation errors and issues when deploying your app to a watchOS 2 device or Watch simulator.

  Workaround: Try the following methods.
  1. If an error sheet listing some FBS error shows up, just run again.
  2. Delete the iOS app an run again.
  3. Ensure that the 'Show App on Apple Watch' is selected in the Apple Watch app for device.
  4. Restart Xcode, the Watch and the iPhone devices/simulators.
  5. Reset the Simulator and Simulator (Watch). To reset the simulators, select the menu, Simulator -> Rest Content and Settings. Then quit them and let Xcode restart them when you Run.
  6. Restart your computer after trying the steps above if necessary.

- The Watch simulator may stop taking input after a reset or reboot. If it doesn't respond to the Home button, quit and restart the Watch simulator application. (21135676)

- If the watch never displayed the trust dialog or don't trust was selected then when you try to Build & Run on the watch, the Debug Navigator will remain empty and not show the '(e) com.xxx.xxx waiting to attach' item.

  Workaround: Disconnecting the iPhone from the Mac and reconnecting it should cause the trust prompt to appear on the Apple Watch. If it still doesn't show then reboot the Apple Watch. A last workaround would be to un-pair the Apple Watch and re-do the pairing with the iPhone.

- When building and running your iOS app containing a WatchKit 2 app, the full WatchKit 2 app may be transferred to your Watch and installed each time even if nothing has changed. (21126483)

- The Debug->Attach To Process menu in Xcode doesn't list processes on the Watch. (20962546)

- When performing the Profile action from Xcode on a WatchKit App the first time, Instruments will show the iOS companion device instead of the watch in the device and process chooser control. Instruments will fail to launch the WatchKit App properly. (21241660)

  Workaround: Launch the Profile action from Xcode again.

- When performing the Profile action from Xcode on a WatchKit App, Instruments may report 'The document "pbxperfconfig.plist" could not be opened." (21161649)

  Workaround: Leave Instruments open, and try running the Profile action from Xcode again. Ensure that the iPhone paired with the Apple Watch is connected to the development OS X system by USB cable.

- When Instruments launches a WatchKit App in System Trace and Time Profiler, Instruments will not get symbols for any system frameworks, instead only displaying addresses. (21259392)

  Workaround: Run the WatchKit App through Xcode, then have Instruments Time Profile attach to the already-running process.

- Storyboard References do not currently work with WatchKit storyboards. (21126778)

- For WatchKit storyboards on OS X 10.11, the glance interface controller attributes inspector may fail to open the popover for selecting a glance template type.

  Workaround: Resize the Xcode window and try again.

- If your home directory is on a case-sensitive filesystem, WatchKit 2 app debugging is not supported. (21123503)

- Building and running as glance will not work if glance isn't enabled in the WatchKit 2 app.

  Workaround: Once you build and run your app for the first time, launch the Apple Watch app and enable the glance for your app. After that, build & run as a glance from Xcode should work.

- When using Xcode or Instruments, there may be times when an Apple Watch device is listed as offline even though the companion iPhone is attached to the computer. (21104615)

  Workaround: Restart Xcode or Instruments.

- Address sanitizer is not supported on watchOS and the watch Simulator (19789677).

- If your home directory is on a case-sensitive filesystem, native watch app debugging is not supported. (21123503)

- When developing on Apple Watch with Xcode, ensure that Automatically Download Apps is enabled under `Apple Watch->General->Automatic Downloads`, so that WatchKit Apps are automatically installed on the Watch and available for debugging, when building/running from Xcode. (21171436)

- Unable to debug WatchKit 1 app extensions in a project that also has WatchKit 2 apps. (21173814)

  Workaround: There isn't currently a way to select the WatchKit 1 app as the one to launch when both exist in the same host iOS app. You need to thin it to the watchKit 1 app for it to work by removing the Watch directory in the host iOS app. Alternately, duplicate the iOS build product and only include the WatchKit 1 app.

General

- Devices running iOS 8.4 beta software are not supported with Xcode 7 beta. (21107693)

- Submitting an archive built with `BITCODE_ENABLED=NO` will fail during upload. (21247555)

  Workaround: Build the archive with `BITCODE_ENABLED=YES` and then uncheck the “Include bitcode” option when distributing the archive (functionally equivalent).

New in Swift 2.0 and Objective-C

Swift Language Features

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  For example:

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

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

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

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

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

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

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

- Enums support multiple generic associated values. (15666173)

For example:

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

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

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

  For example:

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

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

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

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

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

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

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

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

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

Swift Language Changes

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

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

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

  Declaration

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

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

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

  Instead of using bitwise operations such as:

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

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

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

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

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

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

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

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

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

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

  In Swift 1.2:
  ```
  do {
  ...
  } while <condition>
  ```

  In Swift 2.0:
  ```
  repeat {
  ...
  } while <condition>
  ```

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

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

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

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

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

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

  Parameter outline syntax:

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

  Separate parameter fields:

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

  Documenting return values:

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

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

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

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

  This means that code previously written as:

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

  needs to be written as:

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

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

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

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

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

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

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

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

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

Objective-C Language Features

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

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

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

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

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

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

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

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

  For example:

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

  is imported into Swift as

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

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

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

  will be imported into Swift as

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

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

Resolved Issues

Swift 2.0

- Generic type requirements print correctly in the generated module interface for Swift instead of showing up as “T == T”. (19963093)

- Various problems involving protocol conformances not being visible across different Swift files have been fixed (18405215). Additionally, duplicate symbol errors due to protocol conformances being distributed across several Swift files have been eliminated (18448811).

- The compiler produces a warning in cases where a method of a non-`NSObject`-derived class is very close to an optional protocol requirement but does not match because it is not exposed to Objective-C (20219297). For example:

```
class MyController : UIWebViewDelegate {
  func webViewDidStartLoad(v: UIWebView) { … } // warning: non-@objc
method ‘webViewDidStartLoad’ cannot satisfy optional requirement of @objc
protocol ‘UIWebViewDelegate'
}
```

- Adding `@objc(propertyName)` on a Swift property affects the Objective-C name of the property as well as the default Objective-C getter and setter names (19408726). For example:

```
class MyClass : NSObject {
  @objc(theProperty) property: String // Objective-C property is named
“theProperty”
    // Objective-C getter is named “theProperty”
    // Objective-C setter is named “setTheProperty:”
}
```

- The implicit “self” argument to methods is passed as a +0 value instead of a +1 value, leading to significant performance increases for recursive methods (15729033).

Known Issues

Swift Migrator

- "Convert to Latest Swift" may generate build errors when run. These errors can be safely ignored and don't affect the source changes that are produced. (19650497)

- If the migrator is run on code that already migrated to 2.0 and it fails to detect Swift 2.0 syntax, the migrator may perform an unnecessary transformation of a `print(argument)` call to `print(argument, appendNewline: false)`. (20882092)

- Call sites affected by SDK nullablity changes are not transformed comprehensively by the migrator. (20984190)

Swift

- Extended documentation for the Swift standard library is not included in the QuickHelp popover. (21234431)

  View the comments embedded in the Swift standard library.

Swift Language & Compiler

- "`indirect`" enum elements are not implemented yet in this beta, they will be added to a later update.

- A miscompilation or compiler crash may occur when a class inherits an instance of a generic class, overrides its methods, and the overridden method is invoked on the subclass. (20874966)

  For example:

  ```
  class GenericSuper<T> {
    func foo() -> T? {
      return nil
    }
  }

  class GenericSub: GenericSuper<Int> {
    override func foo() -> Int? {
      return 6502
    }
  }

  func foo() {
    GenericSub().foo() // miscompiles
  }
  ```

  Workaround: Mark the override as `final` if appropriate, or coerce the object to the generic superclass before invoking the method.

  ```
  func foo() {
    (GenericSub() as GenericSuper).foo() // OK
  }
  ```

- When extending a Core Foundation type to conform to a protocol, checked casts to the protocol type will fail at runtime in optimized builds. (20882882)

  For example:

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

  import Foundation

  protocol Fooable {
    func foo()
  }

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

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

- Classes that conform to `ErrorType` will cause a runtime crash if you try to bridge them to `NSError` in a `catch` clause. (21116814)

  For example:

  ```
  class MyError: ErrorType {
    let _domain = “”; let _code = 0
  }

  do {
    throw MyError()
  } catch let error as NSError {
    print(error.description)
  }
  ```

  Workaround: Use enums to conform to `ErrorType` instead of classes or make your error classes inherit from `NSError` instead of conforming to `ErrorType`.

- The `guard` statement does not bind variable names properly at the top level of a playground, script file, or in main.swift. (21110580)

- Several global generic functions in the standard library have not yet been moved to be methods.

- Class-constrained protocol types will cause the compiler to crash or emit invalid code if the protocol type is bound to a generic parameter that conforms to `AnyObject.` (18378390)

  For example:

```
func foo<T: AnyObject>(x: T, y: T) {}

protocol CP: class {}
class C: CP {}
class D: CP {}
let x: CP = C(), y: CP = D()
foo(x, y) // T == CP, causes crash
```

Workaround: If possible, use the `AnyObject` type instead of the generic constraint:

```
func foo(x: AnyObject, y: AnyObject) {}
```

or make the protocol `@objc`:

```
@objc protocol CP {}
```

- The compiler may fail to parse a throwing function type in a generic parameter list. (21254633)

  For example:
  ```
  let array = Array<() throws -> ()>(count: 1, repeatedValue: {})
  ```

  Workaround: Declare the function type in a typealias:

  ```
  typealias MyCallback = () throws -> ()
  let array = Array<MyCallback>(count: 1, repeatedValue: {})
  ```