What’s New in Swift

Session 401

Ted Kremenek, Languages & Runtimes Manager
Slava Pestov, Swift Compiler Engineer
Swift open source
Swift 4.2
The Swift Programming Language  [https://swift.org/](https://swift.org/)

<table>
<thead>
<tr>
<th>Branch</th>
<th>Master</th>
<th>New pull request</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Commit</th>
<th>Branch</th>
<th>Title</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>69,785</td>
<td>master</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>736</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>581</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>github</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>apinotes</td>
<td>OS X -&gt; macOS</td>
<td>2 years ago</td>
</tr>
<tr>
<td></td>
<td>benchmark</td>
<td>[benchmark] Fix formatting</td>
<td>4 days ago</td>
</tr>
<tr>
<td></td>
<td>bindings/xml</td>
<td>[Markup] Print Tags in documentation comment XML</td>
<td>11 months ago</td>
</tr>
<tr>
<td></td>
<td>cmake</td>
<td>[cmake] Cleanup the cmake used for adding new fuzzer host tools.</td>
<td>2 days ago</td>
</tr>
<tr>
<td></td>
<td>docs</td>
<td>Merge pull request #15862 from lanzalinux/andoc</td>
<td>2 days ago</td>
</tr>
<tr>
<td></td>
<td>include</td>
<td>Merge pull request #16448 from gottesmann/041f8beaca03cb107c46c3c8b84...</td>
<td>31 minutes ago</td>
</tr>
<tr>
<td></td>
<td>lib</td>
<td>Merge pull request #16457 from nkcsgei/nested-type-init-call</td>
<td>28 minutes ago</td>
</tr>
<tr>
<td></td>
<td>stdlib</td>
<td>Merge pull request #16310 from koher/fast-keys-contains</td>
<td>5 hours ago</td>
</tr>
<tr>
<td></td>
<td>test</td>
<td>Merge pull request #16457 from nkcsgei/nested-type-init-call</td>
<td>28 minutes ago</td>
</tr>
<tr>
<td></td>
<td>tools</td>
<td>[Sema] Error If ObjC Interop is needed when disabled</td>
<td>a day ago</td>
</tr>
<tr>
<td></td>
<td>unitests</td>
<td>libSyntax: add getAbsoluteEndPosition() method to syntax nodes.</td>
<td>7 days ago</td>
</tr>
</tbody>
</table>
600

Code contributors to Swift
Merged pull requests on GitHub

18k
# Swift Releases

## Swift 4.1.2

*Swift 4.1.2 is available as part of Xcode 9.4.*

<table>
<thead>
<tr>
<th>Download</th>
<th>Date</th>
</tr>
</thead>
</table>
| Xcode 9.4*  
(Toolchain) 
(Debugging Symbols) | May 31, 2018 |
| Ubuntu 16.10  
(Signature) | May 31, 2018 |
| Ubuntu 16.04  
(Signature) | May 31, 2018 |
| Ubuntu 14.04  
(Signature) | May 31, 2018 |
Community-Hosted Continuous Integration

- Debian 9.1 on Raspberry Pi
- Fedora 27
- Ubuntu on PowerPC
- Swift for TensorFlow
forums.swift.org
Hello Swift Community,

The review of SE-0202: Random Unification begins now and runs through April 3, 2018.

Reviews are an important part of the Swift evolution process. All reviews should be made in this thread on the Swift forums or, if you would like to keep your feedback private, directly in email to me as the review manager.

The goal of the review process is to improve the proposal under review through constructive criticism and, eventually, determine the direction of Swift. When writing your review, here are some questions you might want to answer in your review:

- What is your evaluation of the proposal?
- Is the problem being addressed significant enough to warrant a change to Swift?
- Does this proposal fit well with the feel and direction of Swift?
- If you have used other languages or libraries with a similar feature, how do you feel that this proposal compares to those?
- How much effort did you put into your review? A glance, a quick reading, or an in-depth study?

More information about the Swift evolution process is available on the Swift Evolution website.

As always, thank you for participating in Swift Evolution.

Ben Cohen
Review Manager
# Introducing: Your Project

I'd like to make some friendly suggestions about how best to introduce your project here. As Ted says in the About thread, this category is here as a service to Swift projects that want an established forum for project ... [read more]

---

**New to Swift and Vapor. Returning data from MySQL**
Tradition suggests that the first program in a new language should print the words “Hello, world!” on the screen. In Swift, this can be done in a single line:

```swift
print("Hello, world!"
```

If you have written code in C or Objective-C, this syntax looks familiar to you—in Swift, this line of code is a complete program. You don’t need to import a separate library for functionality like input/output or string handling. Code written at global scope is used as the entry point for the program, so you don’t need a `main()` function. You also don’t need to write semicolons at the end of every statement.

You can start learning Swift by showing you how to define variables, constants, and functions. If you don’t understand something—everything introduced in this tour is explained in detail in the rest of this book.

**NOTE**

For the best experience, open this chapter as a playground in Xcode. Playgrounds allow you to edit the code listings and see the result immediately.

**Download Playground**

**Simple Values**

Use `let` to define a constant and `var` to define a variable. The value of a constant doesn’t change.
Apple Engagement in the Broader Swift Community

Many Swift-related conferences and podcasts
## Apple Engagement in the Broader Swift Community

<table>
<thead>
<tr>
<th>Event driven networking for Swift</th>
<th>try! Swift Tokyo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementing Swift Generics</td>
<td>LLVM Developers’ Meeting</td>
</tr>
<tr>
<td>Creating Refactoring Transformations for Swift</td>
<td>Swift Summit</td>
</tr>
<tr>
<td>Swift’s Reflective Underpinnings</td>
<td>Swift Summit</td>
</tr>
<tr>
<td>Extending the Standard Library</td>
<td>dotSwift</td>
</tr>
<tr>
<td>Swift 4.1 highlights</td>
<td>Swift Unwrapped</td>
</tr>
</tbody>
</table>
try! Swift San Jose
Friday, June 8
What is Swift 4.2?
Xcode 9.0
Swift 4.0

Xcode 9.3
Swift 4.1

Xcode 10
Swift 4.2
Faster builds

Language features to improve efficiency and remove boilerplate

SDK improvements for Swift

Converging towards binary compatibility
Swift 4.0  Swift 4.1  Swift 4.2
Binary compatibility with future Swift compiler releases

Swift runtime ships in the OS
<table>
<thead>
<tr>
<th>Task</th>
<th>Tracking Issue</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan for the evolution of type metadata (including carving out space for future functionality and freezing performance critical areas)</td>
<td>SR-3923</td>
<td>Done in Swift 4.2</td>
</tr>
<tr>
<td>Clean-up historical artifacts in the metadata representation</td>
<td>SR-3924</td>
<td>Done in Swift 4.2</td>
</tr>
<tr>
<td>Document type metadata with the same representation semantics as object representation semantics</td>
<td>SR-3731</td>
<td></td>
</tr>
<tr>
<td>Decide the mangling of names stored in named value type metadata</td>
<td>SR-3926</td>
<td>Done in Swift 4.2</td>
</tr>
<tr>
<td>Review the efficiency of interacting with the enum discriminator through the witness table</td>
<td>SR-4332</td>
<td>Done in Swift 4.1</td>
</tr>
<tr>
<td>Lock down the layout of value witness tables</td>
<td>SR-3927</td>
<td>Done in Swift 4.1</td>
</tr>
<tr>
<td>Document what class metadata is opaque for library evolution</td>
<td>SR-4343</td>
<td></td>
</tr>
<tr>
<td>Lock down the layout of stables or decide to use thunks</td>
<td>SR-3928</td>
<td></td>
</tr>
</tbody>
</table>

[swift.org/abi-stability](swift.org/abi-stability)
Source Compatibility
Source Compatibility: One Compiler with Three Modes

Swift Compiler - Language
Setting

Swift Language Version
- Swift 3
- Swift 4
- Swift 4.2
- Unspecified
### Source Compatibility: One Compiler with Three Modes

<table>
<thead>
<tr>
<th>Swift 3</th>
<th>Accepts Swift 3 code that built with Xcode 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swift 4</td>
<td>Accepts Swift 4 code that built with Xcode 9.3</td>
</tr>
<tr>
<td>Swift 4.2</td>
<td>Same as with Swift 4 but incorporates new Swift-related SDK improvements</td>
</tr>
</tbody>
</table>
Migrator Support for Swift 4.2

- To Current Swift Syntax...
- To Modern Objective-C Syntax...
- To Objective-C ARC...
Swift 4.2 SDK Changes

Updates to important framework APIs

Some API changes will land in later Xcode 10 betas
Trajectory with Source Compatibility

Source-impacting SDK changes converging

Last release to support “Swift 3” compatibility mode
Faster Swift Debug Builds
Swift Debug Builds

Build speeds of Xcode 10 compared to Xcode 9

- 1.6x Faster build
- 2.4x Faster build
- 2.1x Faster build
1.6x Faster build

Lines of Code

- Headers
- Objective-C
- Swift
1.6x Faster build

Lines of Code:
- Headers
- Objective-C
- Swift

Graph showing the distribution of lines of code.
Speedup for Debug Builds

Exact speed increase varies by project size and number of cores

2x speedup of full builds for many projects

Compilation pipeline optimized to reduce redundant work across files

<table>
<thead>
<tr>
<th>Event</th>
<th>Hall</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building Faster in Xcode</td>
<td>Hall 1</td>
<td>Thursday 9:00AM</td>
</tr>
<tr>
<td>Behind the Scenes of the Xcode Build Process</td>
<td>Hall 2</td>
<td>Friday 2:00PM</td>
</tr>
</tbody>
</table>
## Compilation Mode versus Optimization Level

<table>
<thead>
<tr>
<th>Swift Compiler - Code Generation</th>
<th>Setting</th>
<th>MyApp</th>
</tr>
</thead>
<tbody>
<tr>
<td>▼ Compilation Mode</td>
<td></td>
<td>&lt;Multiple values&gt; ✩</td>
</tr>
<tr>
<td></td>
<td>Debug</td>
<td>Incremental ✩</td>
</tr>
<tr>
<td></td>
<td>Release</td>
<td>Whole Module ✩</td>
</tr>
<tr>
<td>Disable Safety Checks</td>
<td></td>
<td>No ✩</td>
</tr>
<tr>
<td>Exclusive Access to Memory</td>
<td></td>
<td>Full Enforcement (Run-time Checks in Debug Builds Only) ✩</td>
</tr>
<tr>
<td>▼ Optimization Level</td>
<td></td>
<td>&lt;Multiple values&gt; ✩</td>
</tr>
<tr>
<td></td>
<td>Debug</td>
<td>No optimization [-Onone] ✩</td>
</tr>
<tr>
<td></td>
<td>Release</td>
<td>Optimize for Speed [-O] ✩</td>
</tr>
<tr>
<td>Swift 3 @objc Inference</td>
<td></td>
<td>Default ✩</td>
</tr>
</tbody>
</table>
## Compilation Mode versus Optimization Level

<table>
<thead>
<tr>
<th>Setting</th>
<th>MyApp</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compilation Mode</strong></td>
<td></td>
</tr>
<tr>
<td>Debug</td>
<td>Incremental</td>
</tr>
<tr>
<td>Release</td>
<td>Whole Module</td>
</tr>
<tr>
<td><strong>Disable Safety Checks</strong></td>
<td>No</td>
</tr>
<tr>
<td>Exclusive Access to Memory</td>
<td>Full Enforcement (Run-time Checks in Debug Builds Only)</td>
</tr>
<tr>
<td><strong>Optimization Level</strong></td>
<td></td>
</tr>
<tr>
<td>Debug</td>
<td>No optimization [-Onone]</td>
</tr>
<tr>
<td>Release</td>
<td>Optimize for Speed [-O]</td>
</tr>
<tr>
<td>Swift 3 @objc Inference</td>
<td>Default</td>
</tr>
</tbody>
</table>
## Compilation Mode versus Optimization Level

<table>
<thead>
<tr>
<th>Setting</th>
<th>MyApp</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compilation Mode</strong></td>
<td>&lt;Multiple values&gt;</td>
</tr>
<tr>
<td>Debug</td>
<td>Incremental</td>
</tr>
<tr>
<td>Release</td>
<td>Whole Module</td>
</tr>
<tr>
<td>Disable Safety Checks</td>
<td>No</td>
</tr>
<tr>
<td>Exclusive Access to Memory</td>
<td>Full Enforcement (Run-time Checks in Debug Builds Only)</td>
</tr>
<tr>
<td><strong>Optimization Level</strong></td>
<td>&lt;Multiple values&gt;</td>
</tr>
<tr>
<td>Debug</td>
<td>No optimization [-Onone]</td>
</tr>
<tr>
<td>Release</td>
<td>Optimize for Speed [-O]</td>
</tr>
<tr>
<td>Swift 3 @objc Inference</td>
<td>Default</td>
</tr>
</tbody>
</table>
## Compilation Mode versus Optimization Level

<table>
<thead>
<tr>
<th>Setting</th>
<th>MyApp</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compilation Mode</strong></td>
<td>&lt;Multiple values&gt;</td>
</tr>
<tr>
<td>Debug</td>
<td>Incremental</td>
</tr>
<tr>
<td>Release</td>
<td>Whole Module</td>
</tr>
<tr>
<td>Disable Safety Checks</td>
<td>No</td>
</tr>
<tr>
<td>Exclusive Access to Memory</td>
<td>Full Enforcement (Run-time Checks in Debug Builds Only)</td>
</tr>
</tbody>
</table>

| **Optimization Level**           | <Multiple values>          |
| Debug                            | No optimization [-Onone]   |
| Release                          | Optimize for Speed [-O]    |

| Swift 3 @objc Inference          | Default                    |
Stop Using Debug with Whole Module Compilation

Using **Whole Module** for **Debug** builds was a stopgap to improve builds.

**Whole Module** prevents incremental builds.

Use **Incremental** for Debug builds!
Runtime Optimizations
// Calling Convention: "Owned" (+1 retain)

class X { ... }

func caller() {
    // 'x' created with +1 reference count
    let x = X()
    foo(x)
}

func foo(x: X) {
    let y = x.value
    ...
    // release x
}
// Calling Convention: "Owned" (+1 retain)

class X { ... }

func caller() {
    // 'x' created with +1 reference count
    let x = X()
    foo(x)
}

func foo(x: X) {
    let y = x.value
    ...
    // release x
}
// Calling Convention: "Owned" (+1 retain)

class X { ... }

func caller() {
    // 'x' created with +1 reference count
    let x = X()
    // retain x
    // retain x
    bar(x) // release x in callee
    // retain x
    baz(x) // release x in callee
    foo(x) // release x in callee
}

// Calling Convention: "Owned" (+1 retain)

class X { ... }

func caller() {
    // 'x' created with +1 reference count
    let x = X()
    // retain x
    bar(x) // release x in callee
    // retain x
    baz(x) // release x in callee
    foo(x) // release x in callee
}
// Calling Convention: "Owned" (+1 retain)

class X { ... }

func caller() {
    // 'x' created with +1 reference count
    let x = X()
    // retain x
    bar(x) // release x in callee
    // retain x
    baz(x) // release x in callee
    foo(x) // release x in callee
}
// Calling Convention: "Owned" (+1 retain)

class X { ... }

func caller() {
    // 'x' created with +1 reference count
    let x = X()
    // retain x
    bar(x) // release x in callee
    // retain x
    baz(x) // release x in callee
    foo(x) // release x in callee
}
// Calling Convention: "Owned" (+1 retain)

class X { ... }

func caller() {
    // 'x' created with +1 reference count
    let x = X()
    // retain x
    bar(x) // release x in callee
    // retain x
    baz(x) // release x in callee
    foo(x) // release x in callee
}

// Calling Convention: "Guaranteed" (+0 retain)

class X { ... }

func caller() {
    // 'x' created with +1 reference count
    let x = X()
    // retain x
    bar(x) // release x in callee
    // retain x
    baz(x) // release x in callee
    foo(x) // release x in callee
}
class X { ... }

func caller() {
    // 'x' created with +1 reference count
    let x = X()
    // retain x
    bar(x) // release x in callee
    // retain x
    baz(x) // release x in callee
    foo(x) // release x in callee
}

// release x
// Calling Convention: "Guaranteed" (+0 retain)

class X { ... }

func caller() {
    // 'x' created with +1 reference count
    let x = X()
    bar(x)
    baz(x)
    foo(x)
    // release x
}
Small String

Strings now encoded using 16 bytes on 64-bit platforms
Small String

Use 15 bytes to represent the string directly when possible

Sentinel indicates the payload is for a small string

No memory allocations for small strings

Similar to tagged NSString, but can represent slightly larger strings
Reduced Code Size
## Swift Compiler - Code Generation

<table>
<thead>
<tr>
<th>Setting</th>
<th>MyApp</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optimization Level</strong></td>
<td><strong>Optimize for Size [-Osize]</strong></td>
</tr>
<tr>
<td>Debug</td>
<td>Incremental</td>
</tr>
<tr>
<td>Release</td>
<td>Whole Module</td>
</tr>
<tr>
<td>Disable Safety Checks</td>
<td>No</td>
</tr>
<tr>
<td>Exclusive Access to Memory</td>
<td>Full Enforcement</td>
</tr>
<tr>
<td>Swift 3 @objc Inference</td>
<td></td>
</tr>
</tbody>
</table>
-Osize Results on Swift Source Compatibility Suite

Code Size Reduction:

- 0%
- 13%
- 25%
- 38%
- 50%

The chart shows the percentage of code size reduction achieved by the -Osize compiler flag on the Swift source compatibility suite. The y-axis represents the code size reduction percentage, while the x-axis represents the test cases. The bar chart indicates a gradual increase in code size reduction as the tests progress.
Performance Tradeoffs of -Osize

Code size reduced by 10% to 30%

Runtime performance usually 5% slower
New Language Features

Slava Pestov, Swift Compiler Engineer
Swift Evolution Process

Pitch to forums.swift.org

Write draft proposal

Implementation

Review period

Decision by “core team”
apple.github.io/swift-evolution/
## Implemented Proposals

<table>
<thead>
<tr>
<th>Proposal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE-0075</td>
<td>Adding a Build Configuration Import Test</td>
</tr>
<tr>
<td>SE-0143</td>
<td>Conditional Conformances</td>
</tr>
<tr>
<td>SE-0157</td>
<td>Recursive Constraints on Associated Types</td>
</tr>
<tr>
<td>SE-0184</td>
<td>UnsafePointer Improvements</td>
</tr>
<tr>
<td>SE-0185</td>
<td>Synthesizing Equatable and Hashable Conformance</td>
</tr>
<tr>
<td>SE-0186</td>
<td>Remove Ownership Keyword Support in Protocols</td>
</tr>
<tr>
<td>SE-0187</td>
<td>Introduce <code>Sequence.compactMap(_:)</code></td>
</tr>
<tr>
<td>SE-0188</td>
<td>Make Standard Library Index Types Hashable</td>
</tr>
<tr>
<td>SE-0189</td>
<td>Restrict Cross-Module Struct Initializers</td>
</tr>
<tr>
<td>SE-0190</td>
<td>Target Environment Platform Condition</td>
</tr>
<tr>
<td>SE-0191</td>
<td>Eliminate <code>IndexDistance</code> from Collection</td>
</tr>
<tr>
<td>SE-0192</td>
<td>Handling Future Enum Cases</td>
</tr>
<tr>
<td>SE-0193</td>
<td>Cross-Module Inlining and Specialization</td>
</tr>
<tr>
<td>SE-0194</td>
<td>Derived Collection of Enum Cases</td>
</tr>
<tr>
<td>SE-0195</td>
<td>Introduce &quot;Dynamic Member Lookup&quot; Types</td>
</tr>
<tr>
<td>SE-0196</td>
<td>Compiler Diagnostic Directives</td>
</tr>
<tr>
<td>SE-0197</td>
<td>Adding In-Place <code>removeAll(where:)</code></td>
</tr>
<tr>
<td>SE-0198</td>
<td>Playground QuickLook API Revamp</td>
</tr>
<tr>
<td>SE-0199</td>
<td>Adding <code>toggle</code> to <code>Bool</code></td>
</tr>
<tr>
<td>SE-0202</td>
<td>Random Unification</td>
</tr>
<tr>
<td>SE-0204</td>
<td>Add <code>last(where:)</code> and <code>lastIndex(where:)</code> Methods</td>
</tr>
<tr>
<td>SE-0205</td>
<td><code>withUnsafePointer(to:_:)</code> for Immutable Values</td>
</tr>
<tr>
<td>SE-0206</td>
<td>Hashable Enhancements</td>
</tr>
<tr>
<td>SE-0207</td>
<td>Add an <code>allSatisfy</code> Algorithm to <code>Sequence</code></td>
</tr>
<tr>
<td>SE-0210</td>
<td>Add an <code>offset(of:)</code> Method to <code>MemoryLayout</code></td>
</tr>
</tbody>
</table>
## Implemented Proposals

<table>
<thead>
<tr>
<th>Proposal</th>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE-0075</td>
<td>Adding a Build Configuration Import Test</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0143</td>
<td>Conditional Conformances</td>
<td></td>
</tr>
<tr>
<td>SE-0157</td>
<td>Recursive Constraints on Associated Types</td>
<td></td>
</tr>
<tr>
<td>SE-0184</td>
<td>UnsafePointer Improvements</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0185</td>
<td>Synthesizing Equatable and Hashable Conformance</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0186</td>
<td>Remove Ownership Keyword Support in Protocols</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0187</td>
<td>Introduce Sequence.compactMap(::_)</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0188</td>
<td>Make Standard Library Index Types Hashable</td>
<td></td>
</tr>
<tr>
<td>SE-0189</td>
<td>Restrict Cross-Module Struct Initializers</td>
<td></td>
</tr>
<tr>
<td>SE-0190</td>
<td>Target Environment Platform Condition</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0191</td>
<td>Eliminate IndexDistance from Collection</td>
<td></td>
</tr>
<tr>
<td>SE-0192</td>
<td>Handling Future Enum Cases</td>
<td></td>
</tr>
<tr>
<td>SE-0193</td>
<td>Cross-Module Inlining and Specialization</td>
<td></td>
</tr>
<tr>
<td>SE-0194</td>
<td>Derived Collection of Enum Cases</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0195</td>
<td>Introduce &quot;Dynamic Member Lookup&quot; Types</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0196</td>
<td>Compiler Diagnostic Directives</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0197</td>
<td>Adding In-Place removeAll(where:)</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0198</td>
<td>Playground QuickLook API Revamp</td>
<td></td>
</tr>
<tr>
<td>SE-0199</td>
<td>Adding toggle to Bool</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0202</td>
<td>Random Unification</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0204</td>
<td>Add last(where:) and lastIndex(where:) Methods</td>
<td></td>
</tr>
<tr>
<td>SE-0205</td>
<td>withUnsafePointer(to:::) for Immutable Values</td>
<td></td>
</tr>
<tr>
<td>SE-0206</td>
<td>Hashable Enhancements</td>
<td>✔</td>
</tr>
<tr>
<td>SE-0207</td>
<td>Add an allSatisfy Algorithm to Sequence</td>
<td></td>
</tr>
<tr>
<td>SE-0210</td>
<td>Add an offset(of:) Method to MemoryLayout</td>
<td></td>
</tr>
</tbody>
</table>

- Community proposal and implementation
Collection of Enum Cases
// SE-0194 Derived Collection of Enum Cases

definition Gait {
    case walk
    case trot
    case canter
    case gallop
}
// SE-0194 Derived Collection of Enum Cases

enum Gait {
    case walk
    case trot
    case canter
    case gallop
}

    static var allCases: [Gait] = [.walk, .trot, .canter, .gallop]
}

for gait in Gait.allCases {
    print(gait)
}
// SE-0194 Derived Collection of Enum Cases

enum Gait {
    case walk
    case trot
    case canter
    case gallop
    case jog
}

    static var allCases: [Gait] = [.walk, .trot, .canter, .gallop]
}

for gait in Gait.allCases {
    print(gait)
}
// SE-0194 Derived Collection of Enum Cases

class Gait: CaseIterable {  
    case walk  
    case trot  
    case canter  
    case gallop  
    case jog  
}

for gait in Gait.allCases {  
    print(gait)  
}
Conditional Conformance
Inconsistent Behavior in Swift 4.0

```swift
extension Sequence where Element: Equatable {
    func contains(_ element: Element) -> Bool
}
```
Inconsistent Behavior in Swift 4.0

```swift
extension Sequence where Element: Equatable {
    func contains(_ element: Element) -> Bool
}

let animals = 
["cat", "dog", "weasel"]

animals.contains("dog")  // OK
animals.contains("dog")  // OK
```
Inconsistent Behavior in Swift 4.0

```swift
extension Sequence where Element: Equatable {
    func contains(_ element: Element) -> Bool
}

let animals = ["cat", "dog", "weasel"]
animals.contains("dog") // OK

let coins = [[1, 2], [3, 6], [4, 12]]
coins.contains([3, 6]) // Error because the element type [Int] is not equatable
```
Why Aren't All Arrays Equatable?

```swift
let a: [(Int, Int) -> Int] = [{ $0 + $0 }, { 0 - $0 }]
let b: [(Int, Int) -> Int] = [{ $0 * 2 }, { -1 * $0 }]

a == b // Does not make sense
```
extension Array: Equatable where Element: Equatable {
    static func ==(lhs: Array<Element>, rhs: Array<Element>) -> Bool {

    }
}
What If Element Is Equatable?

```swift
extension Array: Equatable where Element: Equatable {
    static func ==(lhs: Array<Element>, rhs: Array<Element>) -> Bool {
        let count = lhs.count
        if count != rhs.count { return false }
        for x in 0..<count {
            if lhs[x] != rhs[x] { return false }
        }
        return true
    }
}
```
let coins = [[1, 2], [3, 6], [4, 12]]
coins.contains([3, 6])  // This now works!
Conditional Conformance

extension Optional: Equatable where Wrapped: Equatable { ... }
extension Array: Equatable where Element: Equatable { ... }
extension Dictionary: Equatable where Value: Equatable { ... }
Conditional Conformance

```swift
extension Optional: Hashable where Wrapped: Hashable { ... }
extension Array: Hashable where Element: Hashable { ... }
extension Dictionary: Hashable where Value: Hashable { ... }
```
Conditional Conformance

```swift
extension Optional: Encodable where Wrapped: Encodable { ... }
extension Array: Encodable where Element: Encodable { ... }
extension Dictionary: Encodable where Key: Encodable, Value: Encodable { ... }
```
Conditional Conformance

extension Optional: Decodable where Wrapped: Decodable { ... }
extension Array: Decodable where Element: Decodable { ... }
extension Dictionary: Decodable where Key: Decodable, Value: Decodable { ... }
let s: Set<[Int?]?> = [[1, nil, 2], [3, 4], [5, nil, nil]]

// Int is Hashable
// => Int? is Hashable
// => [Int?] is Hashable
Synthesized Equatable and Hashable
struct Restaurant {
    let name: String
    let hasTableService: Bool
    let kidFriendly: Bool
}
// SE-0185 Synthesizing Equatable and Hashable Conformance

struct Restaurant {
    let name: String
    let hasTableService: Bool
    let kidFriendly: Bool
}

extension Restaurant: Equatable {
    static func ==(a: Restaurant, b: Restaurant) -> Bool {
        return a.name == b.name &&
            a.hasTableService == b.hasTableService &&
            a.kidFriendly == b.kidFriendly
    }
}
struct Restaurant: Equatable {
    let name: String
    let hasTableService: Bool
    let kidFriendly: Bool
}

// SE-0185 Synthesizing Equatable and Hashable Conformance

struct Restaurant: Hashable {
    let name: String
    let hasTableService: Bool
    let kidFriendly: Bool
}
// Synthesizing Conditional Equatable and Hashable

class Either<Left, Right> {
    case left(Left)
    case right(Right)
}
// Synthesizing Conditional Equatable and Hashable

enum Either<Left, Right> { 
    case left(Left)
    case right(Right)
}

extension Either: Equatable where Left: Equatable, Right: Equatable {
    static func ==(a: Either<Left, Right>, b: Either<Left, Right>) {
        switch (a, b) {
            case (.left(let x), .left(let y)): return x == y
            case (.right(let x), .right(let y)): return x == y
            default: return false
        }
    }
}
// Synthesizing Conditional Equatable and Hashable

class Either<Left, Right> {
    case left(Left)
    case right(Right)
}

extension Either: Equatable where Left: Equatable, Right: Equatable

// Synthesizing Conditional Equatable and Hashable

defined Either<Left, Right> {  
    case left(Left)  
    case right(Right)  
}

extension Either: Equatable where Left: Equatable, Right: Equatable { }  
exension Either: Hashable where Left: Hashable, Right: Hashable { }
// Synthesizing Conditional Equatable and Hashable

enum Either<Left, Right> {
    case left(Left)
    case right(Right)
}

extension Either: Equatable where Left: Equatable, Right: Equatable { }
extension Either: Hashable where Left: Hashable, Right: Hashable { }

// This just works!
var mySet = Set<Either<Int, String>>()
Hashable Enhancements
// Implementing Hashable by Hand in Swift 4.1

struct City {
    let name: String
    let state: String
    let population: Int
}
// Implementing Hashable by Hand in Swift 4.1

struct City {
    let name: String
    let state: String
    let population: Int
}

extension City: Equatable {
    static func ==(a: City, b: City) -> Bool {
        return a.name == b.name && a.state == b.state
    }
}
struct City {
    let name: String
    let state: String
    let population: Int
}

extension City: Hashable {
    var hashValue: Int {
        return name.hashValue ??? state.hashValue
    }
}
// Implementing Hashable by Hand in Swift 4.1

struct City {
    let name: String
    let state: String
    let population: Int
}

extension City: Hashable {
    var hashValue: Int {
        return name.hashValue ^ state.hashValue
    }
}
“Implementing Hashable by Hand in Swift 4.1”

```swift
struct City {
    let name: String
    let state: String
    let population: Int
}

extension City: Hashable {
    var hashValue: Int {
        return (name.hashValue &* 58374501) &+ state.hashValue
    }
}
```
Hash Combining Functions Are Hard

Too much magic

Performance problems

Denial of service attacks

Need a better API!
protocol Hashable {
    var hashValue: Int { get }
}

// Hashable Protocol in Swift 4.1
protocol Hashable {
    func hash(into hasher: inout Hasher)
}
// Using the Hashable Protocol

extension City: Hashable {
    func hash(into hasher: inout Hasher) {
        name.hash(into: &hasher)
        state.hash(into: &hasher)
    }
}

New Hashing Algorithm

Balances hash quality with performance

Random per-process seed
Hash Values Vary From Run to Run

Fix any code that relies on:

• Specific hash values
• Set or Dictionary iteration order
Hash Values Vary From Run to Run

For debugging problems:

- Set `SWIFT_DETERMINISTIC_HASHING=1` in the scheme editor.
Random Number Generation
Random Number Generation in Swift 4.0
Functionality provided through imported C APIs

Requires platform checks

```swift
#if os(iOS) || os(tvOS) || os(watchOS) || os(macOS)
    return Int(arc4random())
#else
    return random() // or Int(rand())
#endif
```
Common operations are tricky

```swift
// Return random number in the range 1...6
func diceRoll() -> Int {
    return 1 + (arc4random() % 6)
}
```

Biased: 1, 2, 3, 4 are more likely than 5, 6
let randomIntFrom0To10 = Int.random(in: 0 ..< 10)
let randomFloat = Float.random(in: 0 ..< 1)
let randomIntFrom0To10 = Int.random(in: 0..<10)
let randomFloat = Float.random(in: 0..<1)

let greetings = ["hey", "hi", "hello", "hola"]
print(greetings.randomElement()!)
let randomIntFrom0To10 = Int.random(in: 0..<10)
let randomFloat = Float.random(in: 0..<1)

let greetings = ["hey", "hi", "hello", "hola"]
print(greetings.randomElement()!!)

let randomlyOrderedGreetings = greetings.shuffled()
print(randomlyOrderedGreetings)
Default RNG Implementation

Secure RNG on both Apple platforms and Linux
// Defining Your Own RNG Algorithm

struct MersenneTwister: RandomNumberGenerator { ... }

var mt = MersenneTwister()
// Defining Your Own RNG Algorithm

struct MersenneTwister: RandomNumberGenerator {
    ...
}

var mt = MersenneTwister()

let randomIntFrom0To10 = Int.random(in: 0..<10, using: &mt)
let randomFloat = Float.random(in: 0..<1, using: &mt)

let greetings = ["hey", "hi", "hello", "hola"]
print(greetings.randomElement()!, using: &mt)

let randomlyOrderedGreetings = greetings.shuffled(using: &mt)
print(greetings)
Checking Platform Conditions
// SE-0075 Adding a Build Configuration Import Test

#if os(iOS) || os(watchOS) || os(tvOS)
    import UIKit
    ...
#else
    import AppKit
    ...
#endif
// SE-0075 Adding a Build Configuration Import Test

#if canImport(UIKit)
  import UIKit
  ...
#else
  import AppKit
  ...
#endif
// SE-0075 Adding a Build Configuration Import Test

#if canImport(UIKit)
    import UIKit
    ...
#endif

#elif canImport(AppKit)
    import AppKit
    ...
#else
    #error("Unsupported platform")
#endif
// SE-0190 Target Environment Platform Condition

#if (os(iOS) || os(watchOS) || os(tvOS)) && (cpu(i386) || cpu(x86_64))
...
#else
   // FIXME: We need to test this better
   ...
#endif
// SE-0190 Target Environment Platform Condition

#if hasTargetEnvironment(simulator)
...
#else
  // FIXME: We need to test this better
  ...
#endif
// SE-0190 Target Environment Platform Condition

#if hasTargetEnvironment(simulator)

...
#else
#warning("We need to test this better")

...
#endif
Implicitly Unwrapped Optionals
Mental Model

Iuo is an attribute of a declaration, not a type of an expression.

First, try type checking value as $T?$ — otherwise, force unwrap to get $T$. 
Optional `Int` can be stored inside `Any`.

Force unwrapping is not performed.

```swift
func computeDangerously(_ b: Bool) -> Int! { return b ? 3 : nil }
func takesAnAny(_ x: Any) { print(x) }

takesAnAny(computeDangerously(.random))
```
Value Type Checks as Type T

Optional Int is not allowed

Must force unwrap the result of the call

```swift
func computeDangerously(_ b: Bool) -> Int! { return b ? 3 : nil }
func takesAnInt(_ x: Int) { print(x) }

takesAnInt(computeDangerously(.random))
```
Value Type Checks as Type `T`

Optional `Int` is not allowed

Must force unwrap the result of the call

```swift
func computeDangerously(_ b: Bool) -> Int! { return b ? 3 : nil }
func takesAnInt(_ x: Int) { print(x) }

takesAnInt(computeDangerously(.random)!)  
```
Corner Cases

IWO not allowed as part of another type:

```swift
let array: [Int!] = [3]
print(array[0])
```

Implicitly unwrapped optionals are only allowed at top level and as function results.
Corner Cases

Not enforced consistently in Swift 4.0:

typealias T = Int!
let array: [T] = [3]
print(array[0])
Corner Cases

IWO in invalid position becomes plain Optional in Swift 4.2:

```swift
typealias T = Int!
let array: [T] = [3]
print(array[0])
```

Using '!' is not allowed here; treating this as '?' instead
Reimplementation of Implicitly Unwrapped Optionals

APRIL 26, 2018
Mark Lacey

A new implementation of implicitly unwrapped optionals (IUOs) landed in the Swift compiler earlier this year and is available to try in recent Swift snapshots. This completes the implementation of SE-0054 - Abolish ImplicitlyUnwrappedOptional Type. This is an important change to the language as it eliminated some inconsistencies in type checking and clarified the rule of how `optional` types are treated.

The main change you’ll see is that diagnostics will now print `??` rather than `??` when referring to a value that was declared as an implicitly unwrapped optional with underlying type `T`. You may also encounter a source compatibility issue that requires you to modify your code before it will compile successfully.

Implicit Unwrapping is Part of a Declaration
Enforcing Exclusive Access to Memory
SE-0176 Enforce Exclusive Access to Memory

Overlapping access to the same memory location not allowed

Combination of static and dynamic checks
/ Example of Exclusive Access Violation

struct Path {
  var components: [String] = []

  mutating func withAppended(_ name: String, _ closure: () -> Void) {
    components.append(name)
    closure()
    components.removeLast()
  }
}
// Example of Exclusive Access Violation

struct Path {
    var components: [String] = []

    mutating func withAppended(_ name: String, _ closure: () -> Void) {
        components.append(name)
        closure()
        components.removeLast()
    }
}

var path = Path(components: ["usr", "local"])
path.withAppended("bin") { print(path) }
// Example of Exclusive Access Violation

struct Path {
    var components: [String] = []

    mutating func withAppended(_ name: String, _ closure: () -> Void) {
        components.append(name)
        closure()
        components.removeLast()
    }
}

var path = Path(components: ["usr", "local"])
path.withAppended("bin") { print(path) }
// Addressing the Exclusive Access Violation

struct Path {
    var components: [String] = []

    mutating func withAppended(_ name: String, _ closure: (Path) -> Void) {
        components.append(name)
        closure(self)
        components.removeLast()
    }
}

var path = Path(components: ["usr", "local"])
path.withAppended("bin", { print($0) })
// More Complex Violation of Exclusive Access

struct Path {
    var components: [String] = []

    mutating func withAppended<T>(_ name: String, _ closure: () -> T) -> T {
        components.append(name)
        let result = closure()
        components.removeLast()

        return result
    }
}
// More Complex Violation of Exclusive Access

```swift
struct Path {
    var components: [String] = []

    mutating func withAppended<T>(_ name: String, _ closure: () -> T) -> T {
        components.append(name)
        let result = closure()
        components.removeLast()
        return result
    }
}

var path = Path(components: ["usr", "local"])
pattern withAppended("bin") { print(path) }
```
// Addressing More Complex Violation of Exclusive Access

struct Path {
    var components: [String] = []

    mutating func withAppended<T>(_ name: String, _ closure: (Path) -> T) -> T {
        components.append(name)
        let result = closure(self)
        components.removeLast()

        return result
    }
}

var path = Path(components: ["usr", "local"])
path.withAppended("bin", { print($0) })
Dynamic Checks Now Available in Release Builds

- Full Enforcement (Run-time Checks in All Builds)
- Full Enforcement (Run-time Checks in Debug Builds Only)
- Compile-time Enforcement
- No Enforcement
Adding `in-place removeAll(where:)`  
Optimize for Size  
Add an `allSatisfy` Algorithm to `Sequence`  
Dynamic Member Lookup  
toggle method on `Bool`  
Small String  
Add an `offset(of:)` Method to `MemoryLayout`  
Refinement of `Exclusive Access`  
#if `hasTargetEnvironment(simulator)`  
Conditional Conformances  
UnsafePointer Improvements  
CaseIterable  
Eliminate `IndexDistance` from `Collection`  
Unification of Random  
Cross-Module Inlining and Specialization  
Faster Incremental Builds  
Refinement of `IUO`  
Hashable Index Types  
#error  
#if `canImport(module)`  
#warning  
Synthesized Equatable and Hashable  
Add last(where:) and lastIndex(where:) Methods  
“Guaranteed” Calling Convention  
Introduce `Sequence.compactMap(_:)`  
Recursive Value Type Metadata with `UnsafePointer(to::)` for Immutable Values
More Information

https://developer.apple.com/wwdc18/401