Documentation Index

This page describes the overall organization of documentation for the Swift toolchain. It is divided into the following sections:

- Tutorials gently guide you towards achieving introductory tasks, while assuming minimal background knowledge.
- How-To Guides help you complete specific tasks in a step-by-step fashion.
- Explanations discuss key subsystems and concepts, at a high level. They also provide background information and talk about design tradeoffs.
- Reference Guides contain a thorough technical reference for complex topics. They assume some overall understanding of surrounding subsystems.
- Recommended Practices suggests guidelines for writing code and diagnostics.
- Project Information tracks continuous integration (CI), branching and release history.
- Evolution Documents includes proposals and manifestos for changes to Swift.
- The External Resources section provides links to valuable discussions about Swift development, in the form of talks and blog posts.
- The Uncategorized section is for documentation which does not fit neatly into any of the above categories. We would like to minimize items in this section; avoid adding new documentation here.

Sometimes documentation is not enough. Especially if you are a new contributor, you might run into roadblocks which are not addressed by the existing documentation. Or they are addressed somewhere but you cannot find the relevant bits. If you are stuck, please use the development category on the Swift forums to ask for help!

Lastly, note that we are slowly moving towards a more structured form of documentation, inspired by the Django project [1] [2]. Hence parts of this page are aspirational and do not reflect how much of the existing documentation is written. Pull requests to clean up the Uncategorized section, or generally fill gaps in the documentation are very welcome. If you would like to make major changes, such as adding entire new pieces of documentation, please create a thread on the Swift forums under the development category to discuss your proposed changes.

Tutorials

- libFuzzerIntegration.md: Using `libFuzzer` to fuzz Swift code.

How-To Guides

- FAQ.md: Answers “How do I do X?” for a variety of common tasks.
- FirstPullRequest.md: Describes how to submit your first pull request. This is the place to start if you’re new to the project!

- GettingStarted.md: Describes how to set up a working Swift development environment for Linux and macOS, and get an edit-build-test-debug loop going.
- DebuggingTheCompiler.md: Describes a variety of techniques for debugging.
- Building for Android:
  - Android.md: How to run some simple programs and the Swift test suite on an Android device.
  - AndroidBuild.md: How to build the Swift SDK for Android on Windows.
- Building for Windows:
  - Windows.md: Overview on how to build Swift for Windows.
  - WindowsBuild.md: How to build Swift on Windows using Visual Studio.
  - WindowsCrossCompile.md: How to cross compile Swift for Windows on a non-Windows host OS.
- Building for OpenBSD:
  - OpenBSD.md: Overview of specific steps for building on OpenBSD.
- RunningIncludeWhatYouUse.md: Describes how to run include-what-you-use on the Swift project.

Explanations

- WebAssembly.md: Explains some decisions that were made while implementing the WebAssembly target.

Compiler and Runtime Subsystems

- Driver:
  - Driver.md: Provides an overview of the driver, compilation model, and the compiler's command-line options. Useful for integration into build systems other than SwiftPM or Xcode.
  - DriverInternals.md: Provides a bird's eye view of the driver's implementation.
- DependencyAnalysis.md: Describes different kinds of dependencies across files in the same module, important for understanding incremental builds.
- DifferentiableProgrammingImplementation.md: Describes how automatic differentiation is implemented in the Swift compiler.
- C and ObjC interoperability: Clang Importer and PrintAsClang
  - CToSwiftNameTranslation.md: Describes how C and ObjC entities are imported into Swift by the Clang Importer.
  - CToSwiftNameTranslation-OmitNeedlessWords.md: Describes how the "Omit Needless Words" algorithm works, making imported names more idiomatic.
- Type-checking and inference:
  - TypeChecker.md: Provides an overview of how type-checking and

inference work.
    - RequestEvaluator.md: Describes the request evaluator architecture, which is used for lazy type-checking and efficient caching.
    - Literals.md: Describes type-checking and inference specifically for literals.
- Serialization.md: Gives an overview of the LLVM bitcode format used for swiftmodules.
    - StableBitcode.md: Describes how to maintain compatibility when changing the serialization format.
- SIL and SIL Optimizations:
    - SILFunctionConventions.md:
    - SILMemoryAccess.md:
    - SILProgrammersManual.md: Provides an overview of the implementation of SIL in the compiler.
    - OptimizerDesign.md: Describes the design of the optimizer pipeline.
    - HighLevelSILOptimizations.rst: Describes how the optimizer understands the semantics of high-level operations on currency data types and optimizes accordingly. Includes a thorough discussion of the `@_semantics` attribute.

SourceKit subsystems

- SwiftLocalRefactoring.md: Describes how refactorings work and how they can be tested.

Language subsystems

- Swift's Object Model
    - LogicalObjects.md: Describes the differences between logical and physical objects and introduces materialization and writeback.
    - MutationModel.rst: Outdated.
- DocumentationComments.md: Describes the format of Swift's documentation markup, including specially-recognized sections.

Stdlib Design

- SequencesAndCollections.rst: Provides background on the design of different collection-related protocols.
- StdlibRationales.rst: Provides rationale for common questions/complaints regarding design decisions in the Swift stdlib.

Reference Guides

- DriverParseableOutput.md: Describes the output format of the driver's `-parseable-output` flag, which is suitable for consumption by editors and IDEs.

- ObjCInterop.md Documents how Swift interoperates with ObjC code and the ObjC runtime.
- LibraryEvolution.rst: Specifies what changes can be made without breaking binary compatibility.
- SIL.rst: Documents the Swift Intermediate Language (SIL).
  - TransparentAttr.md: Documents the semantics of the `@_transparent` attribute.
- DynamicCasting.md: Behavior of the dynamic casting operators `is`, `as?`, and `as!`.
- Runtime.md: Describes the ABI interface to the Swift runtime.
- Lexicon.md: Canonical reference for terminology used throughout the project.
- UnderscoredAttributes.md: Documents semantics for underscored (unstable) attributes.

ABI

- CallConvSummary.rst: A concise summary of the calling conventions used for C/C++, Objective-C and Swift on Apple platforms. Contains references to source documents, where further detail is required.
- CallingConvention.rst: Describes in detail the Swift calling convention.
- GenericSignature.md: Describes what generic signatures are and how they are used in the ABI, including the algorithms for minimization and canonicalization.
- KeyPaths.md: Describes the layout of key path objects (instantiated by the runtime, and therefore not strictly ABI).
  TODO: The layout of key path patterns (emitted by the compiler, to represent key path literals) isn't documented yet.
- Mangling.rst: Describes the stable mangling scheme, which produces unique symbols for ABI-public declarations.
- TypeLayout.rst: Describes the algorithms/strategies for fragile struct and tuple layout; class layout; fragile enum layout; and existential container layout.
- TypeMetadata.rst: Describes the fields, values, and layout of metadata records, which can be used (by reflection and debugger tools) to discover information about types.

Recommended Practices

Coding

- AccessControlInStdlib.rst: Describes the policy for access control modifiers and related naming conventions in the stdlib.
- IndexInvalidation.md: Describes the expected behavior of indexing APIs exposed by the stdlib.
- StdlibAPIGuidelines.rst: Provides guidelines for designing stdlib APIs.

- StandardLibraryProgrammersManual.md: Provides guidelines for working code in the stdlib.
- OptimizationTips.rst: Provides guidelines for writing high-performance Swift code.

Diagnostics

Project Information

- Branches.md: Describes how different branches are setup and what the automerger does.
- ContinuousIntegration.md: Describes the continuous integration setup, including the `@swift_ci` bot.

Evolution Documents

Manifestos

- ABI Stability and Library Evolution
  - ABIStabilityManifesto.md: Describes the goals and design for ABI stability.
  - LibraryEvolutionManifesto.md: Describes the goals and design for Library Evolution.
- BuildManifesto.md: Provides an outline for modularizing the build system for the Swift toolchain.
- CppInteroperabilityManifesto.md: Describes the motivation and design for first-class Swift-C++ interoperability.
- DifferentiableProgramming.md: Outlines a vision and design for first-class differentiable programming in Swift.
- GenericsManifesto.md: Communicates a vision for making the generics system in Swift more powerful.
- OwnershipManifesto.md: Provides a framework for understanding ownership in Swift, and highlights potential future directions for the language.
- StringManifesto.md: Provides a long-term vision for the `String` type.

Proposals

Old proposals are present in the /docs/proposals directory. More recent proposals are located in the apple/swift-evolution repository. You can see the status of different proposals at https://apple.github.io/swift-evolution/.

Surveys

- CallingConvention.rst: This whitepaper discusses the Swift calling convention (high-level semantics; ownership transfer; physical representation; function signature lowering).

- ErrorHandlingRationale.rst: Surveys error-handling in a variety of languages, and describes the rationale behind the design of error handling in Swift.
- WeakReferences.md: Discusses weak references, including the designs in different languages, and proposes changes to Swift (pre-1.0).

Archive

These documents are known to be out-of-date and are superseded by other documentation, primarily The Swift Programming Language (TSPL). They are preserved mostly for historical interest.

- AccessControl.md
- Arrays.rst
- Generics.rst
- ErrorHandling.rst
- StringDesign.rst
- TextFormatting.rst

External Resources

External resources are listed in ExternalResources.md. These cover a variety of topics, such as the design of different aspects of the Swift compiler and runtime and contributing to the project more effectively.

Uncategorized

Needs refactoring

The documents in this section might be worth breaking up into several documents, and linking one document from the other. Breaking up into components will provide greater clarity to contributors wanting to add new documentation.

- ARCOptimization.md: Covers how ARC optimization works, with several examples. TODO: Not clear if this is intended to be an explanation or a reference guide.
- CompilerPerformance.md: Thoroughly discusses different ways of measuring compiler performance and common pitfalls. TODO: Consider breaking up into one high-level explanation explaining key concepts and individual how-to guides that can be expanded independently. Also, it's not the right place to explain details of different compiler modes.
- DevelopmentTips.md: Contains an assortment of tips for better productivity when working on the compiler. TODO: Might be worthwhile to make a dedicated how-to guide or explanation. It might also be valuable to introduce the tips in context, and have the explanation link to all the different tips.
- Diagnostics.md: Describes how to write diagnostic messages and associated educational notes. TODO: Consider splitting into how-tos and

recommended practices. For example, we could have a how-to guide on adding a new diagnostic, and have a recommended practices page which explains the writing style for diagnostics and educational notes.

- HowSwiftImportsCAPIs.md: Contains a thorough description of the mapping between C/ObjC entities and Swift entities. TODO: Not clear if this is intended to be language documentation (for Swift developers), an explanation or a reference guide.
- OptimizerCountersAnalysis.md: TODO: Consider breaking up into a how-to guide on dumping and analyzing the counters and an explanation for the counter collection system.
- Testing.md: TODO: Consider splitting into a how-to guide on writing a new test case and an explanation for how the compiler is tested.
- SwiftIndent.md: TODO: Unclear if this is intended to be an explanation or a reference guide.
- Random.md: Stub.

Archive

- FailableInitializers.rst: Superseded by documentation in The Swift Programming Language.
- InitializerProblems.rst: Describes some complexities around initialization in Swift. TODO: It would be great to have an explanation, say `InitializationModel.md`, that covers the initialization model and new attributes like `@_hasMissingDesignatedInitializers`. Some of this is covered in TSPL's initialization section but that doesn't include newly added attributes.
- Modules.rst: for Swift pre-1.0.
- Swift3Compatibility.md: Discusses the Swift 3 -> Swift 4 migration.
- StoredAndComputedVariables.rst: for Swift pre-1.0.