Swift Migrator

This library implements functionality for the Swift 4 Migrator.

Overview

The Migrator was rewritten from the ground up for Swift 4 with a few major differences:

- It's not a separate tool but integrated directly into the compiler binary
- It understands Swift 3 and Swift 4 code equally
- Can migrate individual targets in Xcode
- A pipeline architecture with explicit immutable state changes

The Migrator runs during a normal frontend invocation, with the *primary file* being the target for migration, resulting in the following additional outputs:

1. The *replacement map* file (a.k.a. *remap* file) `-emit-remap-file-path <path>`
2. The migrated file - optional, primarily for testing. `-emit-migrated-file-path <path>`
3. The migration states - optional, primarily for testing. `-dump-migration-states-dir <dir>`

The majority of changes suggested by the Migrator are driven by:

- API changes from the Xcode 8.3* SDKs and the Xcode 9 SDKs
- Fix-its suggested by the compiler

There are a few passes that walk the AST and perform textual edits for some cases, discussed below.

The Migrator Pipeline

The migrator has the following *passes*, each of which takes an input source text and produces and output source text, collecting a sequence of *states*, which includes the input and output text from the pass.

At the start of the normal frontend invocation, the compiler parses and type-checks the primary input, resulting in a type-checked AST, which is handed to the Migrator if one of the above flags were passed. For this initial step, the Migrator uses whatever Swift language version that was passed to the frontend.

Here are the passes:

1. Pre-fix-it Pass

   If the compiler wasn't able to successfully type-check the primary input source file, the Migrator makes a best effort at applying any fix-its the compiler suggests and trying again, *up to two times*. If it still can't successfully type-check and get an AST, the pipeline stops.

   > *See lib/Migrator/Migrator.cpp:* `Migrator::repeatFixitMigrations`

2. AST Passes

   If the Pre-fix-it Pass was successful, or skipped because it was unnecessary, the *AST Passes* run if you are migrating *from before Swift 4*. These include:

   - API Diff Pass

     This pass injests an *API Diff Data File*, a JSON file describing API changes from the previous SDK, and looks for API references, performing textual edits to update code for things like referenced

type and argument names or optionality changes. This is where the majority of changes come from in the Migrator.

For a list of the different kinds of entries, see `include/swift/IDE/DigesterEnums.def` and the actual JSON data files in `lib/Migrator`.

There are also a few "special case" migrations implemented here, which were different from the typical API diff changes but also rare enough. These are mainly declared in `lib/Migrator/overlay.json`, implemented in `handleSpecialCases`. Some examples of these include:

- Migrating `Double.abs(0.0)` to `Swift.abs(0.0)`
- A few NSOpenGL APIs
- Standard Library `UIntMax` and `IntMax` APIs moving to reference `UInt64` and `Int64`

> *See:*
>
> - *lib/Migrator/APIDiffMigratorPass.cpp*
> - *include/swift/IDE/DigesterEnums.def*
> - *lib/IDE/APIDigesterData.cpp*
> - *lib/Migrator/ios.json*
> - *lib/Migrator/macos.json*
> - *lib/Migrator/tvos.json*
> - *lib/Migrator/watchos.json*

- Tuple Splat Migrator Pass

  This implements a few convenience transformations to ease the transition for SE-0110: Distinguish between single-tuple and multiple-argument function types.

  In particular, this pass adds new variable bindings in closure expressions to destructure what are now are a single argument, a tuple. Prior to SE-0110, a closure's argument list may have been automatically matched up in Swift 3.

  > *See lib/Migrator/RewriteBufferEditsReceiver.cpp*

- type(of:) Migrator Pass

  This is a small convenience pass to account for `Swift.type(of:)` now being resolved by overload resolution. It was handled specially in Swift 3.

  > *See lib/Migrator/Migrator.cpp:* `Migrator::performSyntacticPasses`

3. Post-fix-it Pass

   Finally, the post-fix-it pass, like the pre-fix-it pass, ingests fix-its suggested by the compiler, explicitly set to Swift 4 Mode. This essentially handles automating acceptance of fix-its to convert the source file to Swift 4 in terms of the language itself instead of the APIs.

   This pass is run up to seven times, a number tweaked based on historical observations. The reason the pass is run multiple times is that applying fix-its may reveal more problems to the type-checker, which can then suggest more fix-its, and so on.

> Of note, this includes migration to *Swift 4 @objc Inference*. This is a nuanced topic with implications for your binary size and Objective-C interoperability. There is a link to discussion on this topic at the bottom of this README.

> *See lib/Migrator/Migrator.cpp:* `Migrator::repeatFixitMigrations`

As each of these passes run, a `MigrationState` is pushed onto the Migrator, describing the input and output text explicitly, and which pass produced the transformation.

Finally, at the end of the pipeline, the outputs are emitted. If `-emit-migrated-file-path` was given, the `OutputText` of the final `MigrationState` is written to that file path. If `-dump-migration-states-dir` was specified, the input and output text of each state is dumped into that directory. Finally, if `-emit-remap-file-path` was specified, a file describing the differences between the first and last `MigrationState`'s `OutputText` is emitted.

> *See lib/Migrator/Migrator.cpp:* `swift::migrator::updateCodeAndEmitRemap`

Other controls for the frontend:

- `-disable-migrator-fixits` - skips the fix-it passes during the migration pipeline.
- `-migrate-keep-objc-visibility` - add `@objc` to declarations that were implicitly visible to Objective-C in Swift 3.
- `-api-diff-data-file` - override the API diff JSON file.

> *See include/swift/Migrator/MigratorOptions.h*

Fix-it Filter

For the pre- and post-fix-it passes, there are two basic rules for which fix-its the Migrator will take:

1. Fix-its attached to *error* diagnosics are taken by default and are opt-out.
2. Fix-its attached to *warning* or *note* diagnostics are not taken by default and so are opt-in.

For the opt-out and opt-in cases, these are filtered in the `FixitFilter`, essentially just a small collection of Swift's compiler diagnostic IDs.

> *See include/swift/Migrator/FixitFilter.h*

Applying Fix-its

The `FixitApplyDiagnosticConsumer` delegate class subscribes to fix-its emitted by the type-checker and decides whether to take the fix-it based on the following:

- The fix-it should be for the current primary file of the frontend invocation to prevent multiple fix-its from being emitted for the same file.
- The fix-it should be permitted by the `FixitFilter`.
- The fix-it doesn't suggest a duplicate change to the source file.

In order to produce a `MigrationState` for this pass, fix-its are applied immediately to a running *RewriteBuffer*, which is supplied by Clang. At the end of a fix-it pass, the resulting text is extracted from the `RewriteBufferEditsReceiver`.

> *See:*
>
> - *include/swift/Migrator/FixitApplyDiagnosticConsumer.h*

> - *lib/Migrator/FixitApplyDiagnosticConsumer.cpp*
> - *include/swift/Migrator/RewriteBufferEditsReceiver.h*
> - *lib/Migrator/RewriteBufferEditsReceiver.cpp*

Remap File Format

This is a file describing textual replacements the input file, a JSON array-of-objects. Xcode ingests these files to generate the diff preview you see in the Migration Assistant.

- `file` : String

  The absolute path to the input file.

- `offset` : Number

  The absolute offset into the input file.

- `remove` : Number (Assumed 0 if missing)

  Remove this many bytes at the given `offset` .

- `text` : String (Assumed empty if missing)

  Insert this text at the given `offset` .

These entries can describe *removals*, *insertions*, or *replacements*.

Removals

For removals, you specify `file` , `offset` , and `remove` .

```
{
  "file": "/path/to/my/file.swift",
  "offset": 503,
  "remove": 10
}
```

This says to remove 10 bytes at offset 503 in the original source file. You can specify an empty string for the text.

Insertions

For insertions, you specify `file` , `offset` , and `text` . You can specify `remove` to be 0.

```
{
  "file": "/path/to/my/file.swift",
  "offset": 61,
  "text": ".foo()"
}
```

This says to insert `.foo()` at offset 61 in the source file.

Replacements

For replacements, you specify all four keys.

```
{
  "file": "/path/to/my/file.swift",
  "offset": 61,
  "remove": 3,
  "text": "bar"
}
```

This says to replace the 3 bytes starting at offset 61 with `foo` in the source file.

Other Internals

There are two main other pieces of the Migrator's implementation, *diffing* and *editing*.

Diffing

For diffing, we pulled in an STL port of Google's *diff-match-patch* library to perform the final diff of the start and end `MigrationState` 's text. This is a fairly standard implementation of the Myers Difference Algorithm (see *An O(ND) Difference Algorithm and Its Variations* by Eugene W. Myers).

> *See Diff.h*

Editing

For textual edits during the AST passes, we adapted libEdit's `Commit` and `EditedSource` functionality that was used in the ARC/Objective-C modernizer. Essentially a wrapper that converts Swift's `SourceLoc` , `SourceRange` , and `CharSourceRange` structures into Clang's, this implements basic operations like insertions, removals, and replacements. Originally, we planned on using lib/Syntax for these transformations but there wasn't enough time and we found that the edits we needed for the Migrator were straightforward enough.

> *See:*
>
> - *include/swift/Migrator/EditorAdaptor.h*
> - *lib/Migrator/EditorAdaptor.h*

Migration States

This is an immutable container explicitly describing changes in state as the Migrator runs, which is not only for safety but also for debuggability. This clarifies which pass was responsible for a set of changes in the pipeline because there can be a lot of changes, sometimes conflicting or redundant, between the API diffs and compiler fix-its.

> *See include/swift/Migrator/MigrationState.h*

More Information

Migrating to Swift 4 on swift.org

Migrate to Swift 4 @objc inference on help.apple.com