SourceKit

and You

JP Simard – @simjp – App Builders – Zürich – April 25, 2016

SourceKit *and* Me

JP Simard (🇨🇦 ➡ 🇺🇸 *)*

ObjC/Swift @ Realm

Realm is a mobile database hundreds of millions of people rely on

SourceKit *and* Me

Product Needed API Docs & Clean Code

SourceKit *was the key*

The SourceKit *You* Know

- More than *just* crashes & HUDs
- Very powerful tool

The SourceKit *You* See

The SourceKit *You* See

- Syntax Highlighting
- Code Completion
- Indentation
- Interface Generation
- Documentation Rendering
- Code Structure
- USR Generation

apple/swift/tools/SourceKit

SourceKit is a framework for supporting IDE features like indexing, syntax-coloring, code-completion, etc.

In general it provides the infrastructure that an IDE needs for excellent language support. – *Apple*

The SourceKit *You* Don't See

The SourceKit *You* Don't See

- Components
  - libIDE
  - libParse
  - libFormat
  - libXPC *(Darwin-Only)*
  - libdispatch *(Darwin-Only)*
- C++ Implementation

The SourceKit *You* Interface With

The SourceKit *You* Interface With

- C Interface
- **`sourcekitd.framework`**
- **`libsourcekitdInProc.dylib`**
- Official Python Binding
- Unofficial Swift Binding: SourceKitten

apple/swift/tools/SourceKit

The stable C API for SourceKit is provided via the **`sourcekitd.framework`** which uses an XPC service for process isolation and the **`libsourcekitdInProc.dylib`** library which is in-process. — *Apple*

SourceKitten

An *adorable* little framework and command line tool for interacting with SourceKit.

github.com/jpsim/SourceKitten

```
$ brew install sourcekitten
...
$ sourcekitten version
0.12.2
$ sourcekitten help
Available commands:

   complete    Generate code completion options
   doc         Print Swift docs as JSON or Objective-C docs as XML
   help        Display general or command-specific help
   index       Index Swift file and print as JSON
   structure   Print Swift structure information as JSON
   syntax      Print Swift syntax information as JSON
   version     Display the current version of SourceKitten
```

The SourceKit *You* Build On

The SourceKit *You* Build On

- Code Analysis
- Refactoring
- Migration
- Code Generation

Code Analysis

```
$ cat PublicDeclarations.swift
```

```
public struct A {
  public func b() {}
  private func c() {}
}
```

```
$ echo $query
  [recurse(.["key.substructure"][]?) |
    select(."key.accessibility" == "source.lang.swift.accessibility.public") |
    ."key.name"?]

$ sourcekitten structure --file PublicDeclarations.swift | jq $query
  ["A", "b()"]
```

```
$ cat FunctionsPerStruct.swift

struct A { func one() {} }
struct B { let nope = 0; func two() {}; func three() {} }

$ echo $query
  [."key.substructure"[] | select(."key.kind" == "source.lang.swift.decl.struct") |
    {key: ."key.name", value: [
      (."key.substructure"[] |
        select(."key.kind" == "source.lang.swift.decl.function.method.instance") |
        ."key.name")]}
  ] | from_entries

$ sourcekitten structure --file FunctionsPerStruct.swift | jq $query
  {"A": ["one()"], "B": ["two()", "three()"]}
```

```
$ cat LongFunctionNames.swift

func okThisFunctionMightHaveAnOverlyLongNameThatYouMightWantToRefactor() {}
func nahThisOnesFine() {
  func youCanEvenFindNestedOnesIfYouRecurse() {}
}

$ echo $query
  [recurse(.["key.substructure"][]?) |
      select(."key.kind" | tostring | startswith("source.lang.swift.decl.function")) |
      select((."key.name" | length) > 20) |
      ."key.name"]

$ sourcekitten structure --file LongFunctionNames.swift | jq $query
  [
    "okThisFunctionMightHaveAnOverlyLongNameThatYouMightWantToRefactor()",
    "youCanEvenFindNestedOnesIfYouRecurse()"
  ]
```

Code Refactoring

In ~30 lines of Swift...

Refactoring (`refactor.swift` 1/2)

```
import SourceKittenFramework

let arguments = Process.arguments
let (file, usr, oldName, newName) = (arguments[1], arguments[2], arguments[3], arguments[4])
let index = (Request.Index(file: file).send()["key.entities"] as! [SourceKitRepresentable])
              .map({ $0 as! [String: SourceKitRepresentable] })

func usesOfUSR(usr: String, dictionary: [String: SourceKitRepresentable]) -> [(line: Int, column: Int)] {
    if dictionary["key.usr"] as? String == usr,
        let line = dictionary["key.line"] as? Int64,
        let column = dictionary["key.column"] as? Int64 {
        return [(Int(line - 1), Int(column))]
    }
    return (dictionary["key.entities"] as? [SourceKitRepresentable])?
        .map({ $0 as! [String: SourceKitRepresentable] })
        .flatMap { usesOfUSR(usr, dictionary: $0) } ?? []
}
```

Refactoring (`refactor.swift` 2/2)

```
func renameUSR(usr: String, toName: String) {
    let uses = index.flatMap({ usesOfUSR(usr, dictionary: $0) }).sort(>)
    let fileContents = try! String(contentsOfFile: file)
    var lines = (fileContents as NSString).lines().map({ $0.content })
    for use in uses {
        lines[use.line] = lines[use.line]
          .stringByReplacingOccurrencesOfString(oldName, withString: newName)
    }
    print(lines.joinWithSeparator("\n"))
}

renameUSR(usr, toName: newName)
```

```
$ cat CodeToRefactor.swift
```

```
struct A { let prop = 0 }
struct B { let prop = 1 }
print(A().prop)
print(B().prop)
```

```
$ ./refactor.swift CodeToRefactor.swift s:vV4file1A4propSi prop newProp
```

```
struct A { let newProp = 0 }
struct B { let prop = 1 }
print(A().newProp)
print(B().prop)
```

Code Migration

Same principle as refactoring...

Code Generation

```
$ cat GenerateXCTestManifest.swift

class MyTests: XCTestCase { func nope() {}; func testYolo() {} }

$ echo $query
  [."key.substructure"[] |
    select(."key.kind" == "source.lang.swift.decl.class") |
    select(."key.inheritedtypes"[]."key.name" == "XCTestCase") |
    {key: ."key.name", value: [
      (."key.substructure"[] |
        select(."key.kind" == "source.lang.swift.decl.function.method.instance") |
        select(."key.name" | startswith("test")) |
        ."key.name")]}
  ] | from_entries

$ sourcekitten structure --file GenerateXCTestManifest.swift | jq $query
  {"MyTests": ["testYolo()"]}
```

QueryKit Model Generation (**generate.swift** 1/2)

```
import SourceKittenFramework
struct Property {
    let name: String
    let type: String
    var swiftSourceRepresentation: String {
        return "static let \(name) = Property<\(type)>(name: \"\(name)\")"
    }
}
struct Model {
    let name: String
    let properties: [Property]
    var swiftSourceRepresentation: String {
        return "extension \(name) {\n" +
            properties.map({"  \($0.swiftSourceRepresentation)"}).joinWithSeparator("\n") +
            "\n}"
    }
}
```

QueryKit Model Generation (**generate.swift** 2/2)

```
let structure = Structure(file: File(path: Process.arguments[1])!)
let models = (structure.dictionary["key.substructure"] as! [SourceKitRepresentable]).map({
    $0 as! [String: SourceKitRepresentable]
}).filter({ substructure in
    return SwiftDeclarationKind(rawValue: substructure["key.kind"] as! String) == .Struct
}).map { modelStructure in
    return Model(name: modelStructure["key.name"] as! String,
        properties: (modelStructure["key.substructure"] as! [SourceKitRepresentable]).map({
            $0 as! [String: SourceKitRepresentable]
        }).filter({ substructure in
            return SwiftDeclarationKind(rawValue: substructure["key.kind"] as! String) == .VarInstance
        }).map { Property(name: $0["key.name"] as! String, type: $0["key.typename"] as! String) }
    )
}

print(models.map({ $0.swiftSourceRepresentation }).joinWithSeparator("\n"))
```

```
$ cat QueryKitModels.swift

struct Person {
  let name: String
  let age: Int
}

$ ./generate.swift QueryKitModels.swift

extension Person {
  static let name = Property<String>(name: "name")
  static let age = Property<Int>(name: "age")
}
```

The SourceKit

I Didn't Have Time To Cover

In This Talk

The SourceKit *I Didn't Have Time To Cover* **In This Talk**

- Code Formatting
- Code Completion
- Syntax Highlighting
- Documentation Generation
- More IDE-like Features

The SourceKit

Others

Have Built On

The SourceKit *Others* Have Built On

- **Jazzy**♪♫: Soulful docs for Swift & Objective-C
- **SwiftLint**: A tool to enforce Swift style and conventions
- **Refactorator**: Xcode Plugin that Refactors Swift & Objective-C
- **SourceKittenDaemon**: Swift auto completion for any text editor
- **SwiftKitten**: Swift autocompleter for Sublime Text
- **sourcekittendaemon.vim**: Autocomplete Swift in Vim
- **company-sourcekit**: Emacs company-mode completion for Swift

Next time you need to push Swift further...

Try SourceKit

Thanks!

JP Simard – @simjp – App Builders – Zürich – April 25, 2016