

„Which is which, and who is who?“

**Building a new Swift unqualified name lookup
library during GSoC 2024**

**Jakub Florek
BSc CSE student at TU Delft
www.jakubflorek.com**

01.02.2025 FOSDEM 2025, Brussels

member reference

```
class ArticleReader {  
  var url: URL? = URL(string: "https://example.com/articles.json")  
  var abstracts: [String] = []  
  
  func fetchArticleDescriptions() async {  
    guard let url else { return }  
  
    let data = try? await URLSession.shared.data(from: url).0  
  
    guard let data,  
          let decodedArticles = try? JSONDecoder().decode([Article].self, from: data)  
    else {  
      print("Improper data: \(data?.debugDescription ?? "NO-DATA")")  
      return  
    }  
  
    let abstracts = decodedArticles.map(\.abstract)  
  
    ??? self.abstracts = abstracts  
  
    struct Article: Decodable {  
      let title: String  
      let abstract: String  
    }  
  }  
}
```



reference

shadowing

implicit name

Compiler also needs to understand those relations. It uses **unqualified lookup** to model language's name **semantics**.

SwiftLexicalLookup

Lookup API

- Lightweight, stateless API
- Syntax nodes as entry points
- Returns an array of results
- Enum based data structure
- A result associates extracted names with their source scopes

```
extension SyntaxProtocol {  
    public func lookup(  
        _ identifier: Identifier?,  
        with config: LookupConfig = LookupConfig()  
    ) -> [LookupResult] {  
        scope?.lookup(identifier, at: self.position, with: config) ?? []  
    }  
}
```

```
public enum LookupResult {  
    /// Scope and the names that matched lookup.  
    case fromScope(ScopeSyntax, withNames: [LookupName])  
    /// File scope and names that matched lookup.  
    case fromFileScope(SourceFileSyntax, withNames: [LookupName])  
    /// Names that matched lookup, but are not  
    /// visible in the associated scope.  
    case almostVisible(ScopeSyntax, withNames: [LookupName])  
    /// Indicates where to perform member lookup.  
    case lookInMembers(LookInMembersScopeSyntax)
```

```
public enum LookupName {  
    /// Identifier associated with the name.  
    /// Could be an identifier of a variable, function or closure parameter and more.  
    case identifier(IdentifiableSyntax, accessibleAfter: AbsolutePosition?)  
    /// Declaration associated with the name.  
    /// Could be class, struct, actor, protocol, function and more.  
    case declaration(NamedDeclSyntax)  
    /// Name introduced implicitly by certain syntax nodes.  
    case implicit(ImplicitDecl)
```

```
class ArticleReader {
  var url: URL? = URL(string: "https://example.com/articles.json")
  var abstracts: [String] = []

  func fetchArticleDescriptions() async {
    guard let url else { return }

    let data = try? await URLSession.shared.data(from: url).0

    guard let data,
          let decodedArticles = try? JSONDecoder().decode([Article].self, from: data)
    else {
      print("Improper data: \(data?.debugDescription ?? "NO-DATA")")
      return
    }

    let abstracts = decodedArticles.map(\.abstract)

    self.abstracts = abstracts

    struct Article: Decodable {
      let title: String
      let abstract: String
    }
  }
}
```

Something is missing here...

ClassDeclScope
Flag: lookupMembers

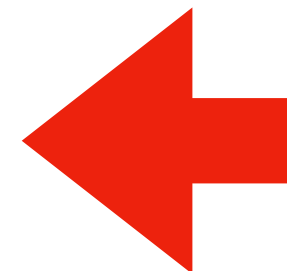
FunctionDeclScope
Implicit: self

GuardStmtScope
Identifier: url

CodeBlockScope
Identifier: data

GuardScope
Identifier: data
Identifier: decoded...

CodeBlockScope
Identifier: abstracts



Unit tests

Cheap and easy to write

- Tests specific scope behavior
- Standard XCTest with a custom reusable harness
- Assertions defined through a custom data structure mimicking the actual expected result

```
func testImplicitErrorInCatchClause() {
    assertLexicalNameLookup(
        source: """
            func foo() {
                let 1error = 0

                do {
                    try x.bar()
                    2error
                } 6catch SomeError {
                    3error
                } 4catch {
                    5error
                }
            }
            """,
        references: [
            "2": [
                .fromScope(CodeBlockSyntax.self, expectedNames: [
                    NameExpectation.identifier("1")
                ])
            ],
            "3": [
                .fromScope(CodeBlockSyntax.self, expectedNames: [
                    NameExpectation.identifier("1")
                ])
            ],
            "5": [
                .fromScope(CatchClauseSyntax.self, expectedNames: [
                    NameExpectation.implicit(.error("4"))
                ]),
                .fromScope(CodeBlockSyntax.self, expectedNames: [
                    NameExpectation.identifier("1")
                ])
            ],
        ],
    )
}
```

```
class ArticleReader {
    var url: URL? = URL(string: "https://example.com/articles.json")
    var abstracts: [String] = []

    func fetchArticleDescriptions() async {

        guard let url else { return }

        let data = try? await URLSession.shared.data(from: url).0

        guard let data,
              let decodedArticles = try? JSONDecoder().decode([Article].self, from: data)
        else {
            print("Improper data: \(data?.debugDescription ?? "NO-DATA")")
            return
        }

        let abstracts = decodedArticles.map(\.abstract)

        self.abstracts = abstracts

        struct Article: Decodable {
            let title: String
            let abstract: String
        }
    }
}
```

```
class ArticleReader {
  func fetchArticleDescriptions() async {

    guard let url else { return }

    let data = try? await URLSession.shared.data(from: url).0

    guard let data,
          let decodedArticles = try? JSONDecoder().decode([Article].self, from: data)
    else { return }

    let abstracts = decodedArticles.map(\.abstract)

    self.abstracts = abstracts
  }
}
```

```
func testExample() {
  assertLexicalNameLookup(
    source: """
      class ArticleReader {
        func fetchArticleDescriptions() async {

          guard let url else { return }

          let data = try? await URLSession.shared.data(from: url).0

          guard let data,
                let decodedArticles = try? JSONDecoder().decode([Article].self, from: data)
          else { return }

          let abstracts = decodedArticles.map(\\.abstract)

          self.abstracts = abstracts
        }
      }
    """
  )
}
```

```
func testExample() {
  assertLexicalNameLookup(
    source: """
      class ArticleReader {
        7func fetchArticleDescriptions() async {

          guard let 6url else { return }

          let 5data = try? await URLSession.shared.data(from: url).0

          guard let 4data,
                let 3decodedArticles = try? JSONDecoder().decode([Article].self, from: data)
          else { return }

          let 2abstracts = decodedArticles.map(\\.abstract)

          self.abstracts = 1abstracts
        }
      }
    """
  )
}
```

```
func testExample() {
  assertLexicalNameLookup(
    source: """
      class ArticleReader {
        7func fetchArticleDescriptions() async {

          guard let 6url else { return }

          let 5data = try? await URLSession.shared.data(from: url).0

          guard let 4data,
            let 3decodedArticles = try? JSONDecoder().decode([Article].self, from: data)
          else { return }

          let 2abstracts = decodedArticles.map(\\.abstract)

          self.abstracts = 1abstracts
        }
      }
    """,
    references: [
      "1": [],
    ]
  )
}
```

```

func testExample() {
  assertLexicalNameLookup(
    source: """
      class ArticleReader {
        7func fetchArticleDescriptions() async {

          guard let 6url else { return }

          let 5data = try? await URLSession.shared.data(from: url).0

          guard let 4data,
                let 3decodedArticles = try? JSONDecoder().decode([Article].self, from: data)
          else { return }

          let 2abstracts = decodedArticles.map(\\.abstract)

          self.abstracts = 1abstracts
        }
      }
    """,
    references: [
      "1": [
        .fromScope(CodeBlockSyntax.self, expectedNames: ["2"]),
        .fromScope(GuardStmtSyntax.self, expectedNames: ["3", "4"]),
        .fromScope(CodeBlockSyntax.self, expectedNames: ["5"]),
        .fromScope(GuardStmtSyntax.self, expectedNames: ["6"]),
        .fromScope(FunctionDeclSyntax.self, expectedNames: [NameExpectation.implicit(.self("7"))]),
        .lookInMembers(ClassDeclSyntax.self),
      ],
    ],
    useNilAsTheParameter: true
  )
}

```

Integration tests

Run inside the compiler

- Used to ensure equality of both implementations
- Easy to run for real world code
- Validates SLL against the compiler
- Produces glanceable tables
- Accounts for some funky compiler behavior...

```
-----> Lookup started at: 1706:7 ("debugPrint") finishInSequentialScope: false
> | ASTScope | SwiftLexicalLookup
> ✓ | k 1699:10 | k 1699:10
> ✓ | v 1699:13 | v 1699:13
> ✗ | first 1698:9 | type 1690:18
> ✗ | result 1697:9 | self 1689:17
> ✗ | type 1690:18 | ↕ V 1689:49
> ✗ | self 1689:17 | ↕ K 1689:46
> ⓘ | Omitted SwiftLexicalLookup name: Self 1683:1
> ✗ | V 1689:49 | Look memb: 1684:11
> ✗ | K 1689:46 | -----
> ⓘ | Omitted ASTScope name: K 1689:46
> ⓘ | Omitted ASTScope name: V 1689:49
> ✗ | End here: Look memb: 1684:11 | -----
```

Interesting cases...

Sequential scopes

```
class ArticleReader {
  var url: URL? = URL(string: "https://example.com/articles.json")
  var abstracts: [String] = []

  func fetchArticleDescriptions() async {

    guard let url else { return }
    let data = try? await URLSession.shared.data(from: url).0
    guard let data,
          let decodedArticles = try? JSONDecoder().decode([Article].self, from: data)
    else {
      print("Improper data: \(data?.debugDescription ?? "NO-DATA")")
      return
    }

    let abstracts = decodedArticles.map(\.abstract)

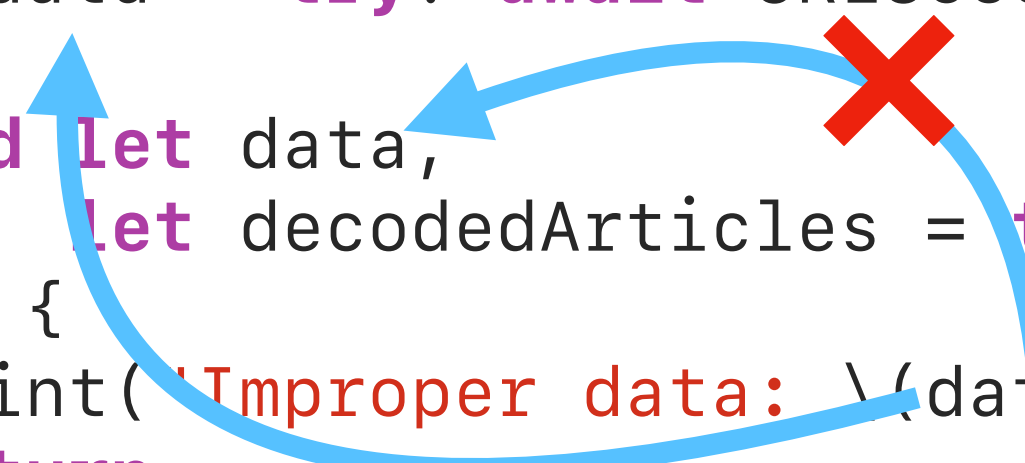
    self.abstracts = abstracts

    struct Article: Decodable {
      let title: String
      let abstract: String
    }
  }
}
```

Names introduced in guard are visible for consecutive conditions.

Sequential scopes

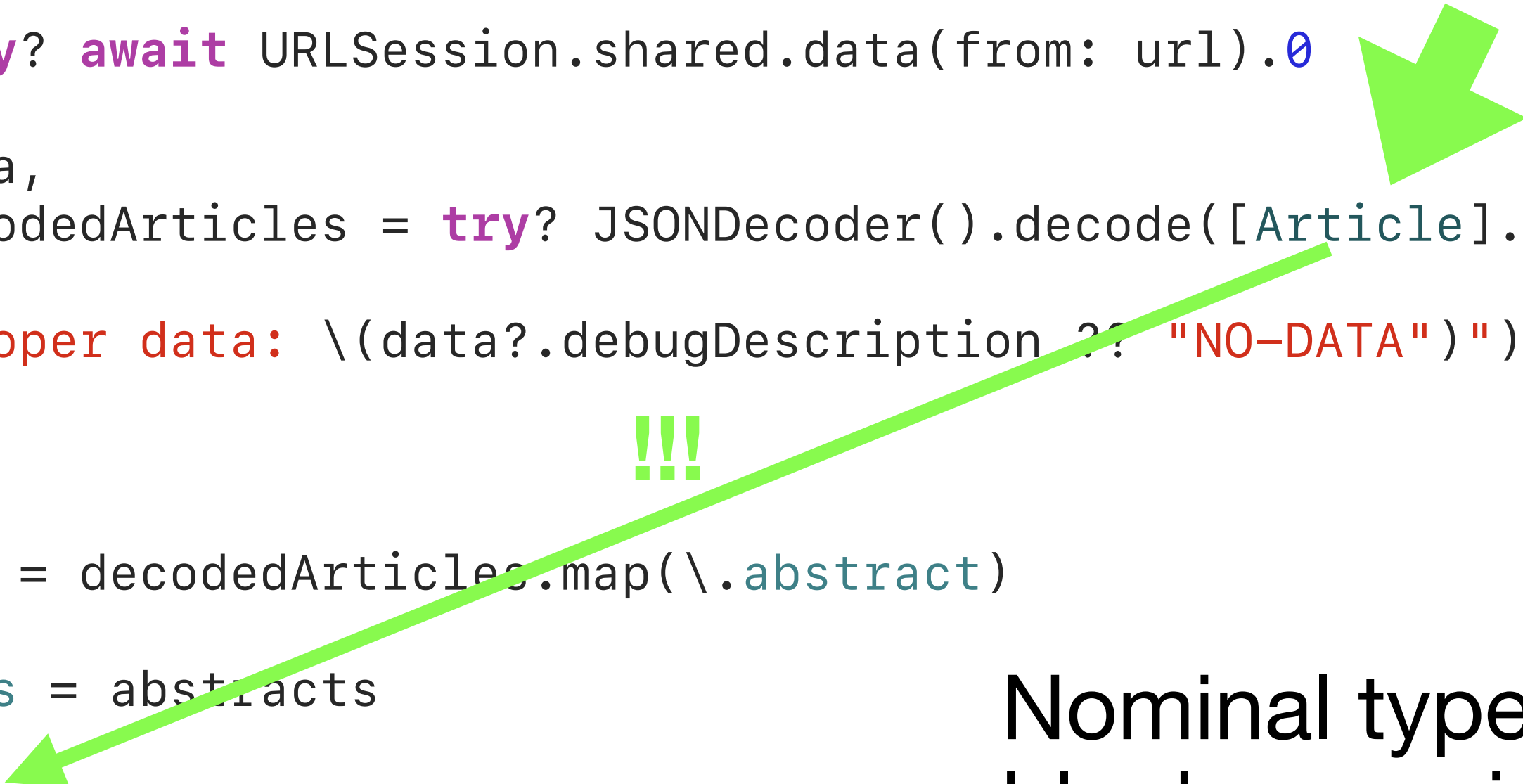
```
class ArticleReader {  
  var url: URL? = URL(string: "https://example.com/articles.json")  
  var abstracts: [String] = []  
  
  func fetchArticleDescriptions() async {  
  
    guard let url else { return }  
  
    let data = try? await URLSession.shared.data(from: url).0  
  
    guard let data,  
          let decodedArticles = try? JSONDecoder().decode([Article].self, from: data)  
    else {  
      print("Improper data: \(data?.debugDescription ?? "NO-DATA")")  
      return  
    }  
  
    let abstracts = decodedArticles.map(\.abstract)  
  
    self.abstracts = abstracts  
  
    struct Article: Decodable {  
      let title: String  
      let abstract: String  
    }  
  }  
}
```



Introduced names are not visible
inside the guard scope body.

Sequential scopes

```
class ArticleReader {  
  var url: URL? = URL(string: "https://example.com/articles.json")  
  var abstracts: [String] = []  
  
  func fetchArticleDescriptions() async {  
  
    guard let url else { return }  
  
    let data = try? await URLSession.shared.data(from: url).0  
  
    guard let data,  
          let decodedArticles = try? JSONDecoder().decode([Article].self, from: data)  
    else {  
      print("Improper data: \(data?.debugDescription ?? "NO-DATA")")  
      return  
    }  
  
    let abstracts = decodedArticles.map(\.abstract)  
  
    self.abstracts = abstracts  
  
    struct Article: Decodable {  
      let title: String  
      let abstract: String  
    }  
  }  
}
```



The diagram illustrates the visibility of nominal types across sequential scopes. A green arrow points from the `Article` struct declaration inside the `fetchArticleDescriptions` function to its use in the `decode` call within the same function. Another green arrow points from the `Article` struct declaration to its use in the `self.abstracts` assignment. A green exclamation mark is placed between the struct declaration and the assignment, indicating that the struct is visible in the scope where it is used.

Nominal types declared in code blocks are visible across the whole scope body.

Macro declarations

A bug on the compiler side

- Compiler incorrectly first introduces generic parameters before function parameters
- Issue #77141
- Exemplifies the hard process of bringing the two implementations together

```
@freestanding(expression)
public macro externalMacro<T>(
    module: String,
    type: String
) -> T = Builtin.ExternalMacro
```

```
-----> Lookup started at: 4:9 ("String") finishInSequentialScope: false
>  | ASTScope | SwiftLexicalLookup
>  | T 5:6 | module 3:3
>  | Omitted ASTScope name: T 5:6
>  | module 3:3 | type 4:3
>  | type 4:3 | T 5:6
```

did I
How ~~to~~ contribute to Swift?

First contribution roadmap

- Pre-GSoC confusion
- Experimenting with swift-syntax
- Building a demo
- Writing the proposal

- Initial planning
- Developing the library in isolation
- Unit testing
- Test Driven Development

- Working with the main repo
- Exponential growth of complexity
- Testing against the compiler
- Validating the implementation

- Stabilizing the api
- Improving performance
- Publishing RFC last week
- Building functionality around SLL

Thank you



[linkedin.com/in/jakubfl](https://www.linkedin.com/in/jakubfl)



github.com/MAJKFL

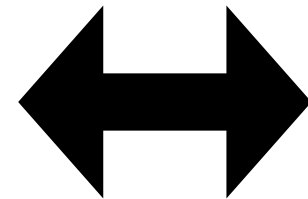


kubaflor23@gmail.com

Equivalent names

```
var state = LoadingState.notStarted
```

```
func getLoadingDetails() -> String? {  
  switch state {  
  case .success(let details):  
    return details  
  case .failure(let details):  
    return details  
  case .running:  
    return nil  
  case .notStarted:  
    return nil  
  }  
}
```



```
var state = LoadingState.notStarted
```

```
func getLoadingDetails() -> String? {  
  switch state {  
  case .success(let details), .failure(let details):  
    return details  
  case .running, .notStarted:  
    return nil  
  }  
}
```

!!!

A name recursively associated with **multiple** names with the **same semantic meaning** and the first one acting as the **representative**.

Helpful links

Jump start your contributions!

- QuickStart guide: <https://www.swift.org/quickstart-contribution/>
- Contributing to swift-syntax: <https://github.com/swiftlang/swift-syntax/blob/main/CONTRIBUTING.md>
- Contributing to the swift main repository: <https://github.com/swiftlang/swift/blob/main/CONTRIBUTING.md>
- Getting started with the main Swift repository: <https://github.com/swiftlang/swift/blob/main/docs/HowToGuides/GettingStarted.md>
- Tool for visualizing swift syntax tree: <https://swift-ast-explorer.com/>

Further reading

- Deeper dive in SwiftLexicalLookup: <https://forums.swift.org/t/gsoc-2024-swiftlexicallookup-a-new-lexical-name-lookup-library/75889>
- SwiftLexicalLookup documentation: <https://swiftpackageindex.com/swiftlang/swift-syntax/main/documentation/swiftlexicallookup>
- Issue #77141 „Wrong result order in ASTScope::unqualifiedLookup for macro declarations.”: <https://github.com/swiftlang/swift/issues/77141>
- GSoC 2024 work product submission: <https://www.jakubflorek.com/posts/gsoc-2024>