USETECH

Проще, чище, быстрее. Макросы Swift

Анна Жаркова
Lead Mobile developer

Обо мне

- В мобильной разработке с 2013
- Ведущий мобильный разработчик в Usetech
- Нативная разработка под iOS и Android (Swift/Objective-C, Kotlin/Java) кросс-платформа (Xamarin, Kotlin multiplatform)
- Ментор, управляю командой направления
- Спикер на конференциях AppsConf, Mobius, TechTrain, DroidCon (2022)
- Преподаватель в Otus (iOS Pro и базовый)
- Автор статей по мобильной разработке (SwiftUI, iOS, KMM)
- Эксперт Skillbox

mb Experts

USETECH

Обсудим:

- Макросы Swift 5.9 . Концепция
- Анатомия макросов
- Расширяем типы. Async/await
- iOS Retrofit на макросах
- Генерируем UI. Макросы в SwiftUI

Макросы Swift 5.9. WWDC 2023

Расширения для компилятора Swift

Макросы Swift 5.9. WWDC 2023

- SE-0382 • Expression Macros
- SE-0389 • Attached Macros
- SE-0397 • Freestanding Declaration Macros
- SE-0394 • Package Manager Support for Custom Macros

Макросы Swift. Назначение

- Преобразование существующего кода
- Генерация нового кода
- Упрощение и ускорение разработки

Макросы Swift. Назначение

- Преобразование существующего кода
- Генерация нового кода
- **Упрощение и ускорение разработки**
- **Сокращение написания кода разработчиком**

Кодогенерация Kotlin

- Шаблоны на плагинах IDE
- Annotation Processing (KSP, KAPT)
- Kotlin Compiler Plugins

Кодогенерация Swift

- Шаблоны Xcode IDE
- SwiftGen, скрипты
- Swift Macros

Старые макросы iOS

- Objective-C, макросы препроцессора
- В Swift через Bridge
- Результат при вызове в рантайме

```
///.h
#define AD_SIZE CGSizeMake(320, 50)
+ (CGSize)adSize;

/// .m
+ (CGSize)adSize { return AD_SIZE; }
```

Старые макросы iOS

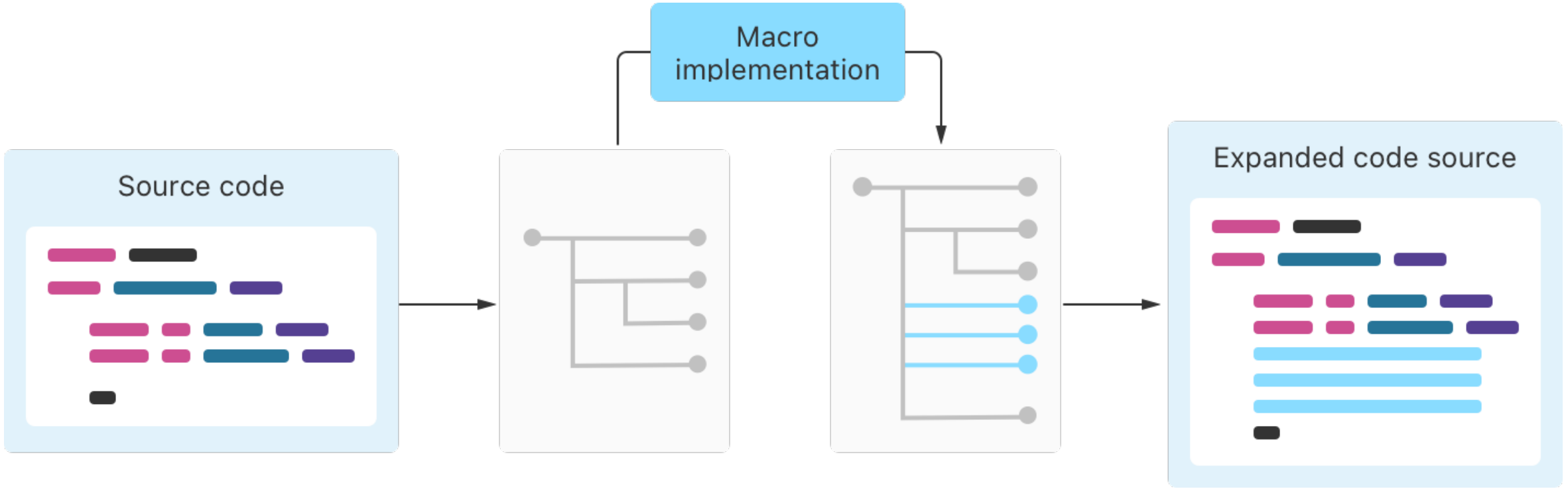
- Objective-C, макросы препроцессора
- В Swift через Bridge
- Результат при вызове в рантайме

```
///.h
#define AD_SIZE CGSizeMake(320, 50)
+ (CGSize)adSize;

/// .m
+ (CGSize)adSize { return AD_SIZE; }
```

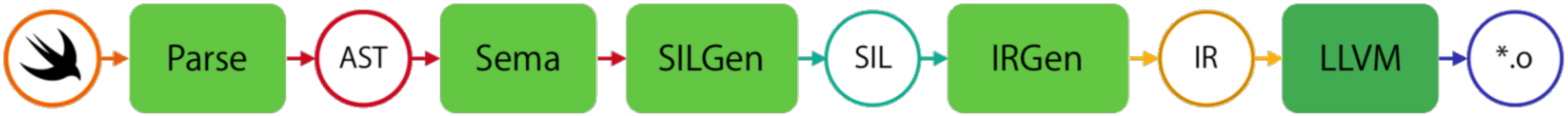
Это другое!!!

Макросы Swift 5.9. Расширение



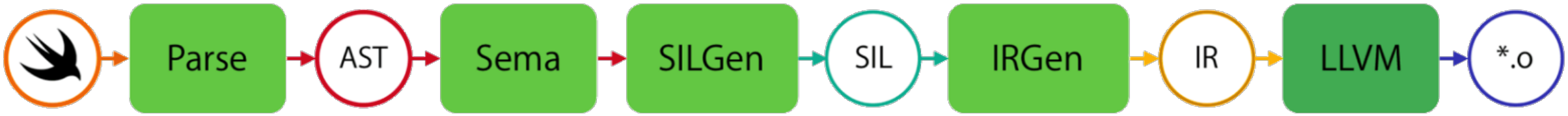
Заглянем под капот

Swift. Компиляция

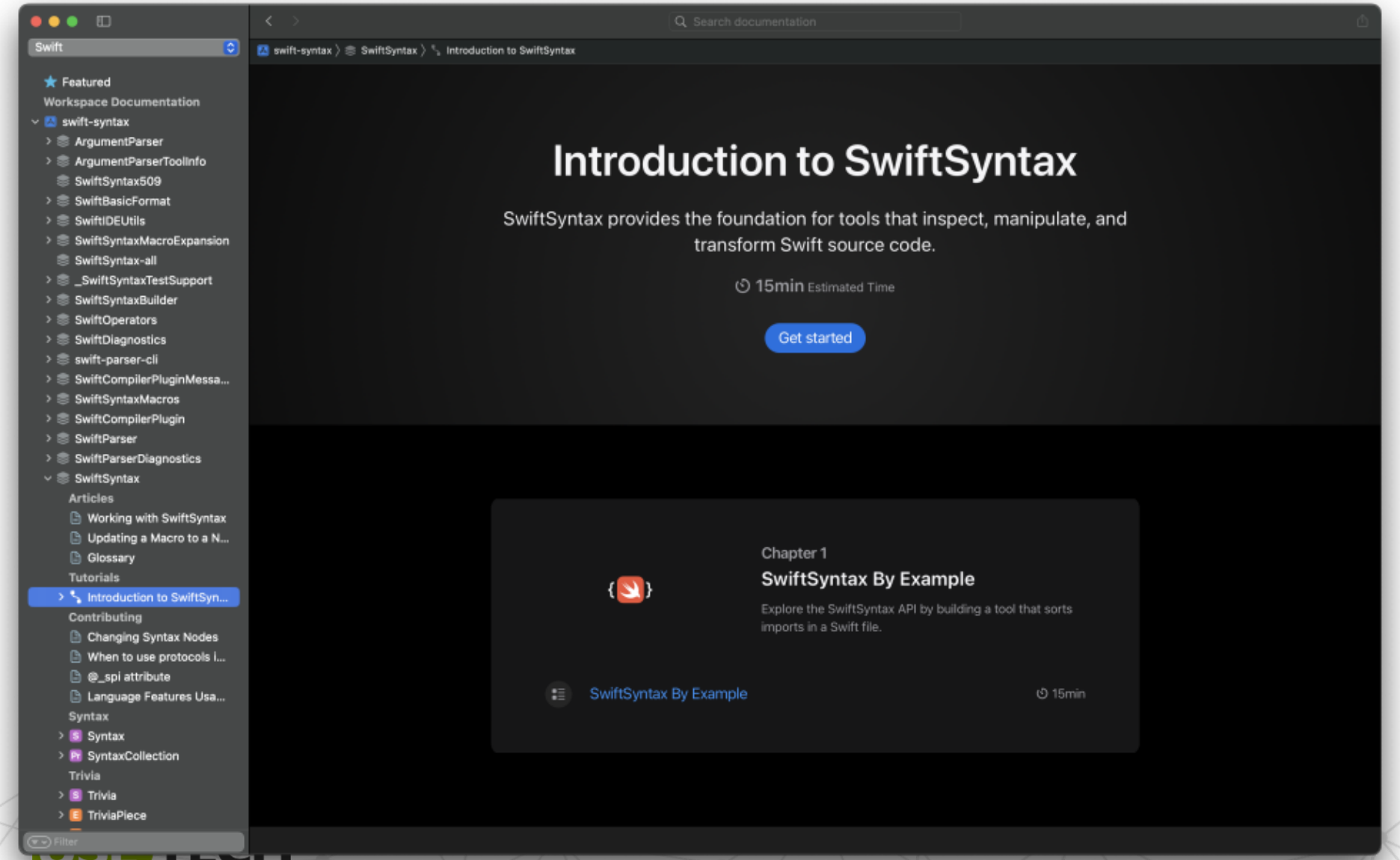


Swift. Компиляция

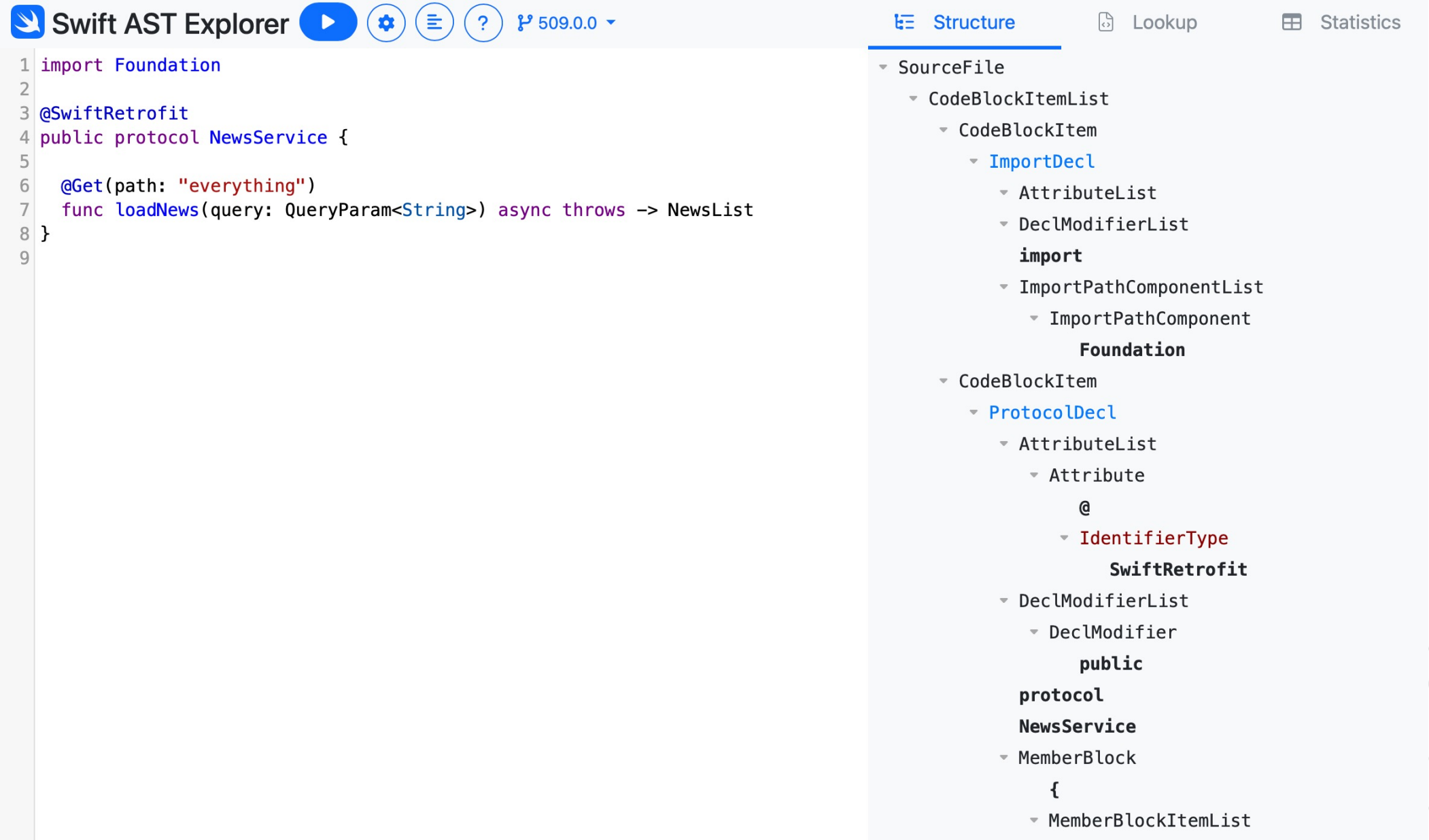
Swift Intermediate Language



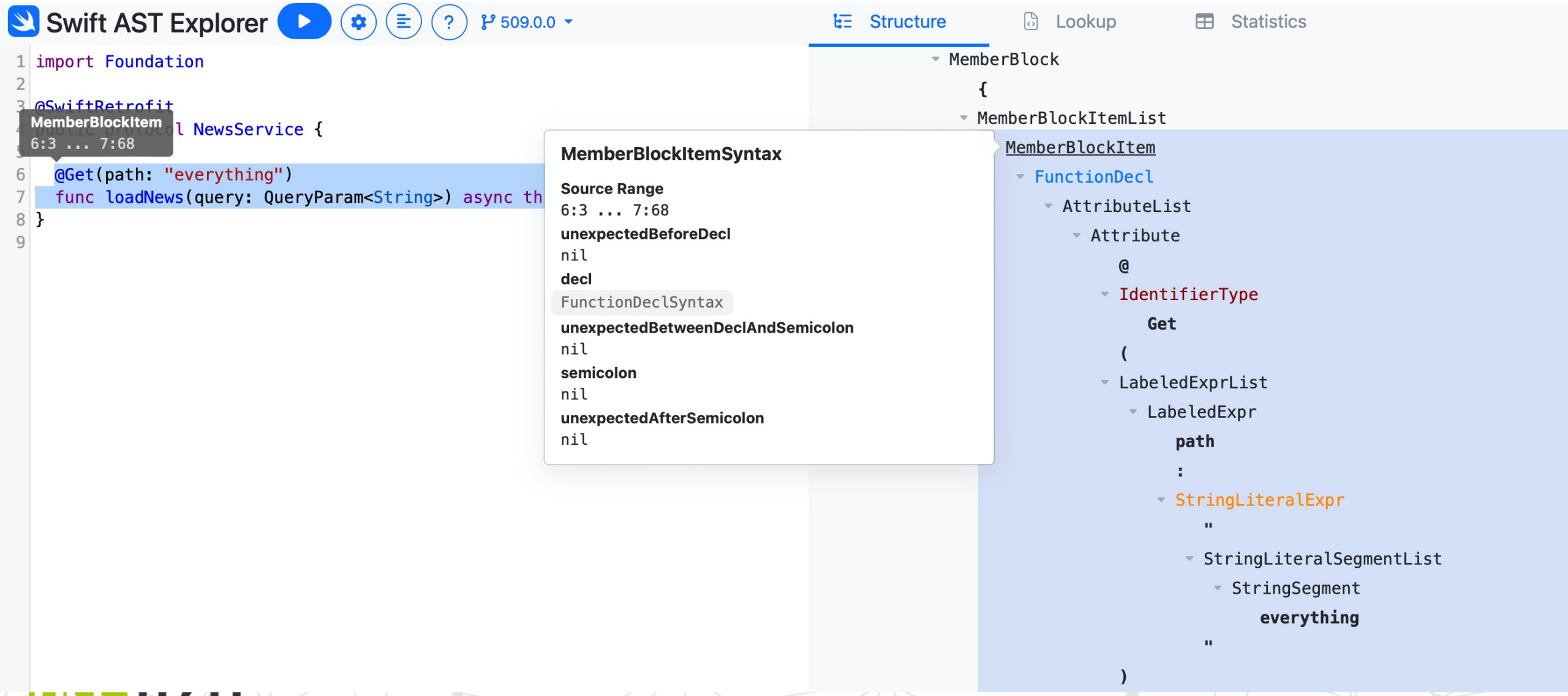
SwiftSyntax



Swift AST


Swift AST Explorer
509.0.0
Structure
Lookup
Statistics
1 import Foundation
2
3 @SwiftRetrofit
4 public protocol NewsService {
5
6 @Get(path: "everything")
7 func loadNews(query: QueryParam<String>) async throws -> NewsList
8 }
9
SourceFile
CodeBlockItemList
CodeBlockItem
ImportDecl
AttributeList
DeclModifierList
import
ImportPathComponentList
ImportPathComponent
Foundation
CodeBlockItem
ProtocolDecl
AttributeList
Attribute
@
IdentifierType
SwiftRetrofit
DeclModifierList
DeclModifier
public
protocol
NewsService
MemberBlock
{
MemberBlockItemList

Swift AST

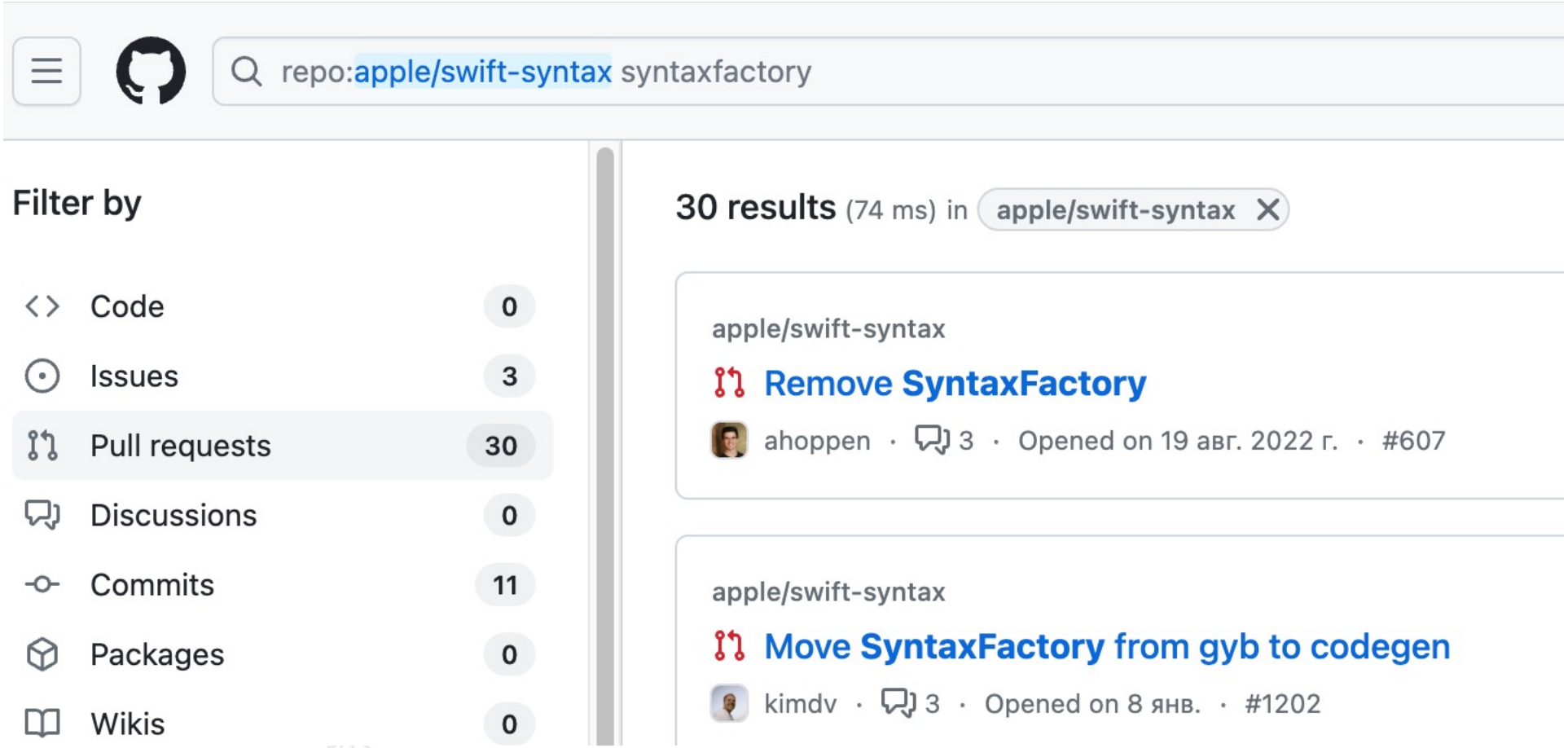

Swift AST Explorer
509.0.0
Structure
Lookup
Statistics
1 import Foundation
2
3 @SwiftRetrofit
4 public protocol NewsService {
5
6 @Get(path: "everything")
7 func loadNews(query: QueryParam<String>) async th
8 }
9
MemberBlockItem
6:3 ... 7:68
MemberBlockItemSyntax
Source Range
6:3 ... 7:68
unexpectedBeforeDecl
nil
decl
FunctionDeclSyntax
unexpectedBetweenDeclAndSemicolon
nil
semicolon
nil
unexpectedAfterSemicolon
nil
MemberBlock
{
MemberBlockItemList
MemberBlockItem
FunctionDecl
AttributeList
Attribute
@
IdentifierType
Get
(
LabeledExprList
LabeledExpr
path
:
StringLiteralExpr
"
StringLiteralSegmentList
StringSegment
everything
"
)

SyntaxFactory

```
//struct Example {}
let structKeyword = SyntaxFactory.makeStructKeyword(trailingTrivia: .spaces(1))
let identifier = SyntaxFactory.makeIdentifier("Example", trailingTrivia: .spaces(1))

let leftBrace = SyntaxFactory.makeLeftBraceToken()
let rightBrace = SyntaxFactory.makeRightBraceToken(leadingTrivia: .newlines(1))
let members = MemberDeclBlockSyntax { builder in
    builder.useLeftBrace(leftBrace)
    builder.useRightBrace(rightBrace)
}
let structureDeclaration = StructDeclSyntax { builder in
    builder.useStructKeyword(structKeyword)
    builder.useIdentifier(identifier)
    builder.useMembers(members)
}
```

SyntaxFactory, удален из SwiftSyntax



SyntaxRewriter

SyntaxRewriter

SyntaxVisitor

```
public class ZalgoRewriter: SyntaxRewriter {
    public override func visit(_ token: TokenSyntax) -> Syntax {
        guard case let .stringLiteral(text) = token.tokenKind else {
            return token
        }

        return token.withKind(.stringLiteral(zalgo(text)))
    }
}
```

SyntaxRewriter

Применение Rewriter

Займемся макросами

Виды макросов

- Автономные @freestanding
- Прикрепленные @attached

Автономные макросы. @freestanding

Дополняют объявление или значение выражения. Не меняют исходное объявление

- @freestanding(expression)
- @freestanding(declaration)

Автономные макросы. @freestanding

Дополняют объявление или значение выражения. Не меняют исходное объявление

```
let image = #unwrap(request.downloadedImage, message: "was already checked")

// Begin expansion for "#unwrap"
{ [wrappedValue = request.downloadedImage] in
    guard let wrappedValue else {
        preconditionFailure(
            "Unexpectedly found nil: ‘request.downloadedImage’ " +
                "was already checked",
            file: "main/ImageLoader.swift",
            line: 42
        )
    }
    return wrappedValue
}()
// End expansion for "#unwrap"
```

Автономные макросы. @freestanding

Дополняют объявление или значение выражения. Не меняют исходное объявление

```
let image = #unwrap(request.downloadedImage, message: "was already checked")

        // Begin expansion for "#unwrap"
        { [wrappedValue = request.downloadedImage] in
            guard let wrappedValue else {
                preconditionFailure(
                    "Unexpectedly found nil: ‘request.downloadedImage’ " +
                        "was already checked",
                    file: "main/ImageLoader.swift",
                    line: 42
                )
            }
            return wrappedValue
        }()
        // End expansion for "#unwrap"
```

Автономные макросы. @freestanding

Дополняют объявление или значение выражения. Не меняют исходное объявление

```
/// Force-unwraps the optional value passed to `expr`.
/// - Parameter message: Failure message, followed by `expr` in single quotes
@freestanding(expression)
macro unwrap<Wrapped>(_ expr: Wrapped?, message: String) -> Wrapped

let image = #unwrap(downloadedImage, message: "was already checked")
```

Прикрепленные. @attached

- Изменяют объявление, к которому они прикреплены
- Начинаются с @

@attached != PropertyWrappers

- Макросы выполняются во время компиляции
- PropertyWrappers выполняются в Runtime
- Их можно использовать вместе

@attached. Роли

- attached(peer)
- attached(accessor)
- attached(memberAttribute)
- attached(member)
- attached(conformance)

@attached. Роли

- attached(peer)
- attached(accessor)
- attached(memberAttribute)
- attached(member)
- **attached(conformance) -> extension**

@attached. Роли

Роли можно комбинировать

Дает макросу доступ к разным элементам кода

```
@attached(member, names: arbitrary)
@attached(conformance)
@attached(peer, names: prefixed(SwiftRetrofit))
public macro SwiftRetrofit() = #externalMacro(module: "SwifRetrofitMacros",
                                              type: "SwiftRetrofitMacro")
```

Ограничения макросов

- Макросы не видят расширения друг друга
- Каждый макрос работает с оригинальным кодом
- Область применения макроса ограничена

Ограничения макросов

- Макросы не видят расширения друг друга
- Каждый макрос работает с оригинальным кодом
- Область применения макроса ограничена

Пустые макросы можно заменить PropertyWrapper или alias

Ограничения макросов

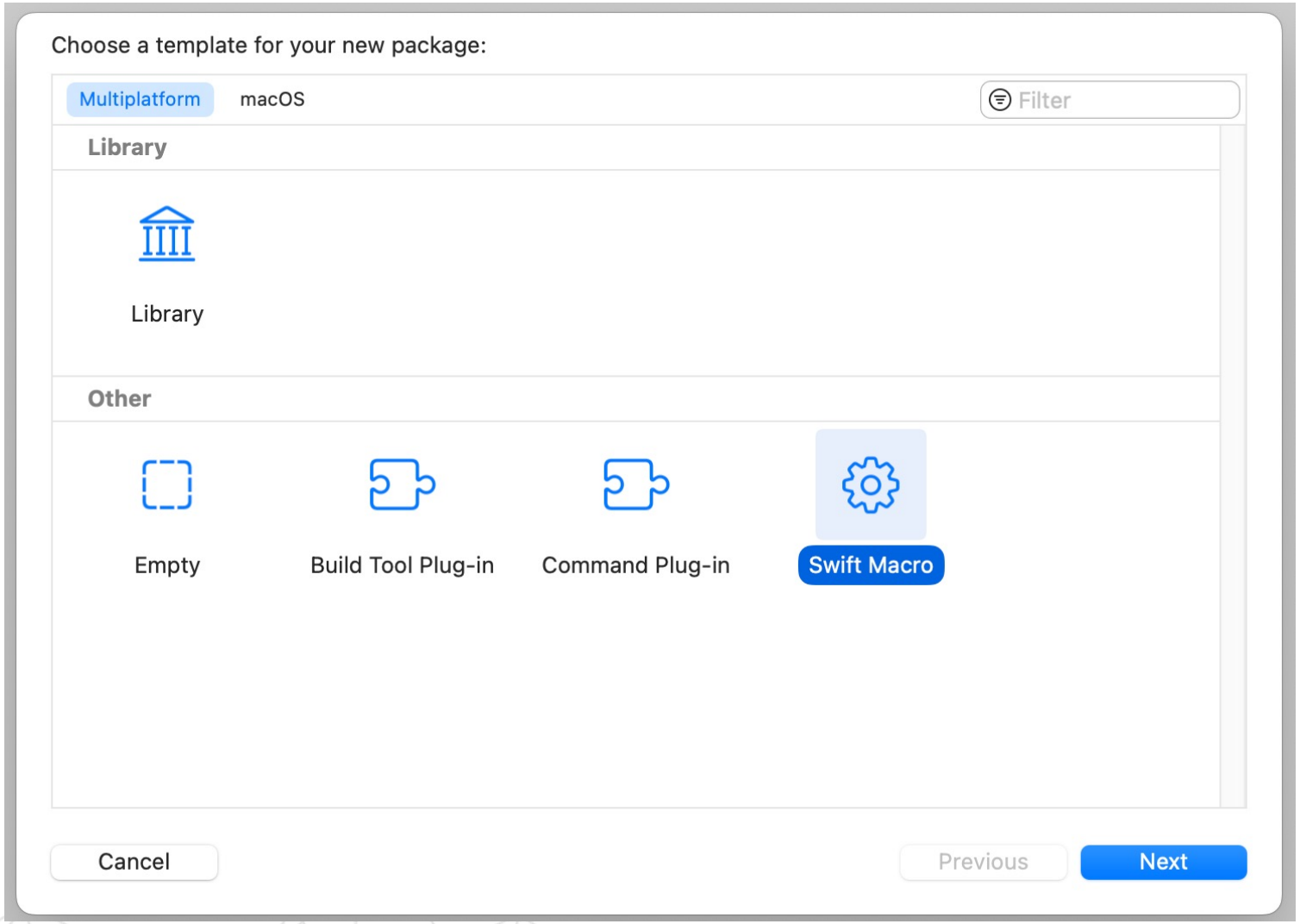
- Макросы не видят расширения друг друга
- Каждый макрос работает с оригинальным кодом
- Область применения макроса ограничена

Не работает как KSP или плагины Kotlin

Пустые макросы можно заменить PropertyWrapper или alias

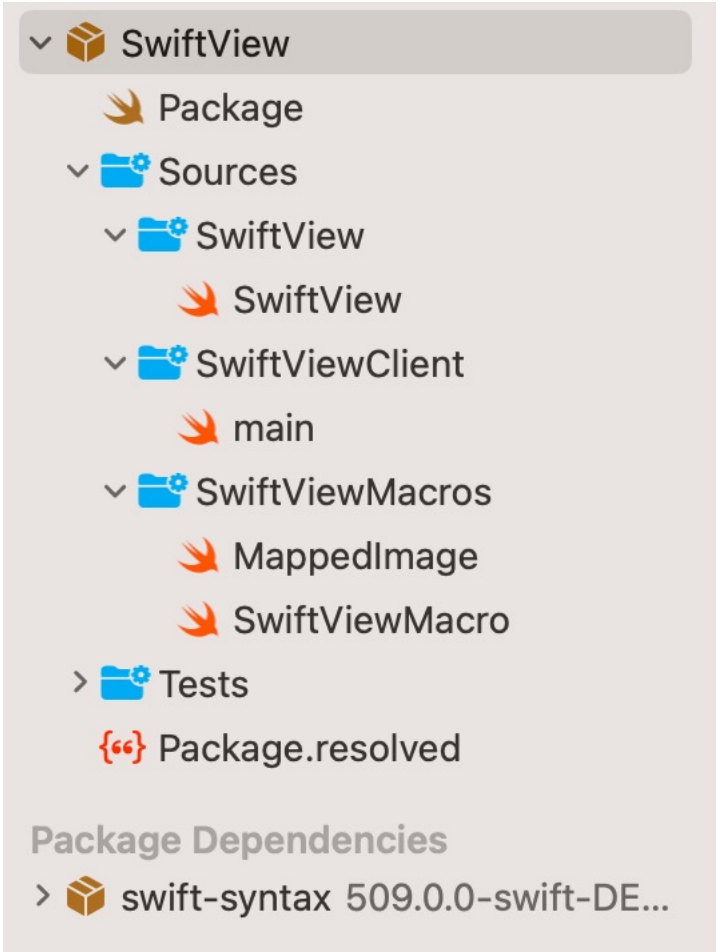
Разберем создание макроса

Создание макроса



Создание макроса

- Макрос
- Плагин компилятора Swift
- Тестовое окружение



AddAsync. CompletionHandler -> Async/await

Генерация перегрузки метода с оберткой async/await

```
@AddAsync
  func fetch(completion: (Data) -> Void) {
      completion(Data())
}
###unwrapped
func fetch() async -> Data {
    await withCheckedContinuation { continuation in
      fetch { value in
          continuation.resume(with: .success(value))
      }
    }
}
```

Создание макроса. AddAsync

Декларация макроса

Выбираем роль

@attached(peer), PeerMacro

```
@attached(peer, names: overloaded)
public macro AddAsync() =
        #externalMacro(module: "SwiftTestMacros",
        type: "AsyncPeerMacro")
```

Создание макроса. AddAsync

Плагин компилятора

```
@main
struct AsyncPeerPlugin: CompilerPlugin {
    let providingMacros: [Macro.Type] = [
        AsyncPeerMacro.self
    ]
}
```

Создание макроса.AddAsync

Структура типа макроса. Body ???

```
public struct AsyncPeerMacro: PeerMacro {
    public static func expansion(of node: AttributeSyntax,
                                 providingPeersOf declaration: some DeclSyntaxProtocol,
                                 in context: some MacroExpansionContext) throws -> [DeclSyntax] {

        guard let function = declaration.as(FunctionDeclSyntax.self) else {
            throw AsyncDeclError.onlyApplicableToFunction
        }

        let body = createBody()

        return [DeclSyntax(stringLiteral: body)]
    }
}
```

AddAsync. План-капкан

- Проверить, что декларация является функцией
- Взять параметр типа closure
- Извлечь данные
- Создать новую декларацию из шаблона и данных
- Profit!

Создание макроса.AddAsync

functionCompletionParameter

```
guard function.signature.parameterClause.parameters.count == 1,

        let functionCompletionParameter = function
        .signature.parameterClause
        .parameters.first?.type.as(FunctionTypeSyntax.self),

        let functionCompletionParameterType = functionCompletionParameter
        .parameters.first?.type.as(IdentifierTypeSyntax.self)

        else {

    throw error
}
```

Создание макроса.AddAsync

Заполним шаблон

```
///functionCompletionParameterType
/// function
let body = """
        func \(function.name.text)() async -> \(functionCompletionParameterType.name) {
            await withCheckedContinuation { continuation in
                \(function.name.text) { value in
                    continuation.resume(with: .success(value))
                }
            }
        }
        """
```

Создание макроса. Расширение

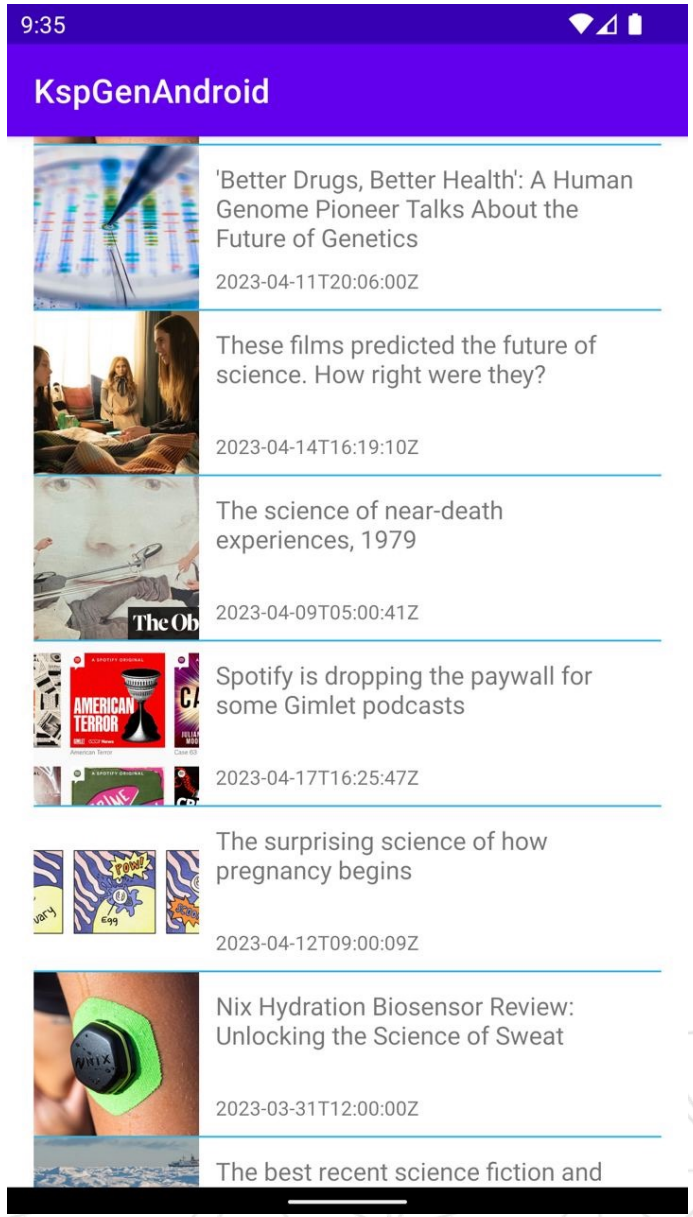
Моментальный результат

```
18  @AddAsync
19    func fetch(completion: (Data) -> Void) {
20        completion(Data())
21    }
      func fetch() async -> Data {                    @AddAsync
2         await withCheckedContinuation { continuation in
3             fetch { value in
4                 continuation.resume(with:
                      .success(value))
5             }
6         }
7     }
```

А теперь реальный псевдо-боевой кейс

KSP. Composable экран с логикой

Готовый сэмпл
https://github.com/anioutkazharkova/kgen-android



Спикеры

Анна Жаркова
Usetech

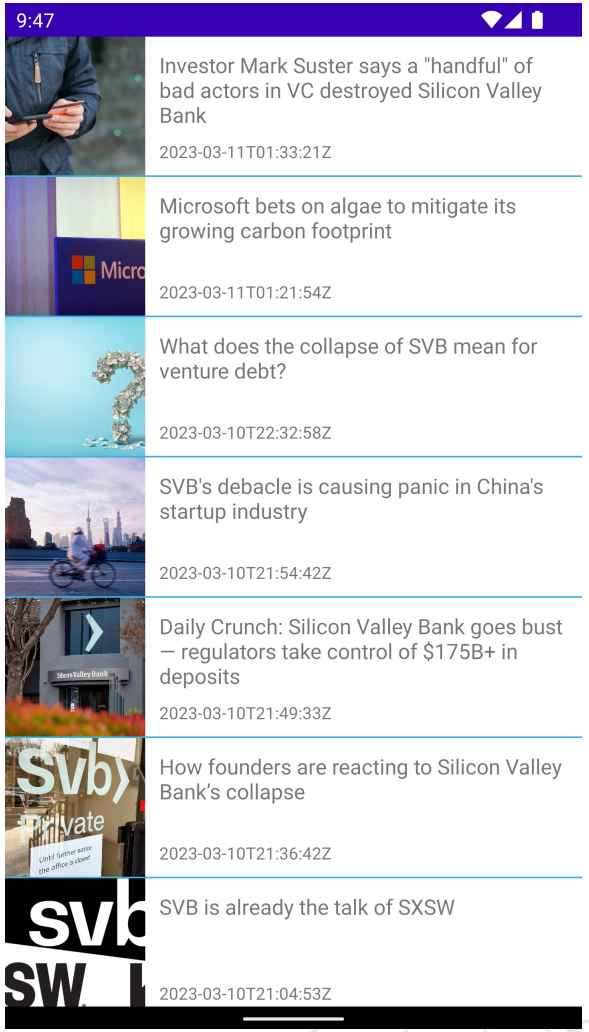
ДОКЛАД | UI/UX в мобильной разработке | 19.05 / 18:45 – 19:30 (UTC+7)

Упрощаем и укрощаем UI для Android с помощью аннотаций

Kotlin compiler plugins

Готовый сэмпл

https://github.com/anioutkazharkova/ksp-kmm-cases



Спикеры

Анна Жаркова
Usetech

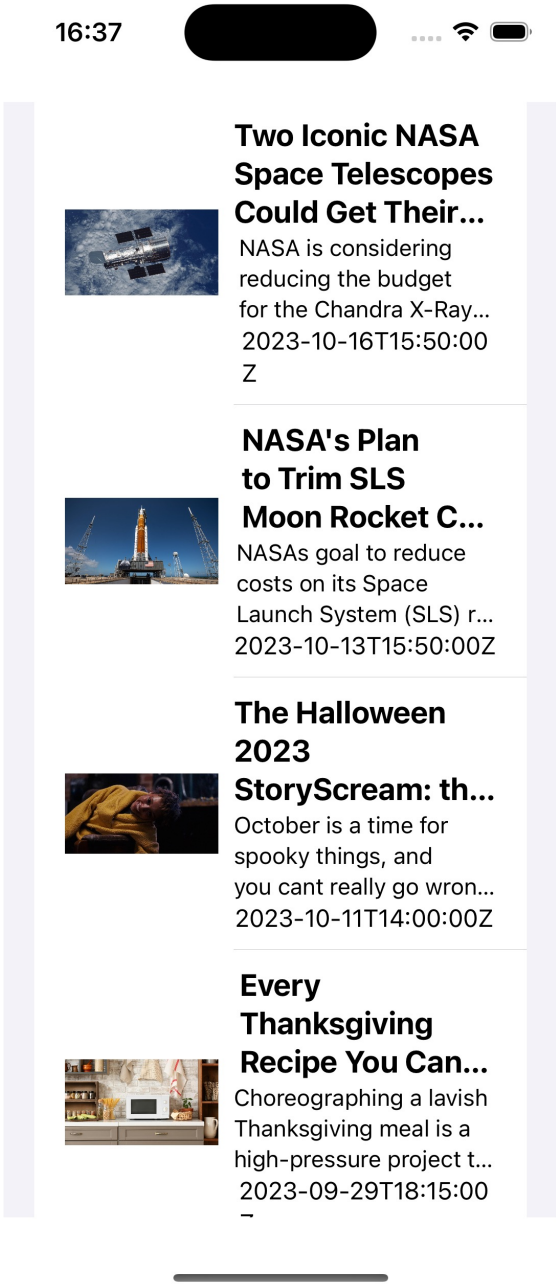
ДОКЛАД Под капотом 12.05 / 20:15 – 21:00 (UTC+7)

Оно само: используем плагины компиляции

Что будем делать

Список новостей:

- Экран SwiftUI на макросах
- Сетевой клиент в стиле ретрофит на макросах



Свой ретрофит для Swift

USETECH

Swift Retrofit

```
9  import SwifRetrofit
10 import SwiftSyntax
11
12 @SwiftRetrofit
13 public protocol NewsService {
14
15     @Get(path: "everything")
16     func loadNews(query: QueryParam<String>)async throws ->NewsList
17
18 }
19
20
```

Line: 20 Col: 1

Swift Retrofit

- @SwiftRetrofit
- @Get
- @Query

```
@SwiftRetrofit
public protocol NewsService {

    @Get(path: "everything")
    func loadNews(@Query("q") query: String)async throws ->NewsList

}
```

Swift Retrofit. Встроенный сетевой клиент

NetworkClient – сетевой клиент

Исходник лежит в плагине

Создается на стороне клиентского приложения

```
public class SwiftRetrofitNewsService: NewsService, NetworkServiceProtocol {

    public static func createInstance(client: NetworkClient) ->
        any NewsService {
            return SwiftRetrofitNewsService(client: client)
        }

    private let client: NetworkClient

    private init(client: NetworkClient) {
        self.client = client
    }
    //Далее Magic код
    //
    //...
}
```

Swift Retrofit. Встроенный сетевой клиент

NetworkClient – сетевой клиент

Исходник лежит в плагине

Создается на стороне клиентского приложения

```
class DI {
    static let shared = DI()

    lazy var networkClient: NetworkClient = {
        return NetworkClient.Builder(config: NetworkConfiguration())
            .build()
    }()

    lazy var newsService: NewsService? = {
        return SwiftRetrofitNewsService
            .createInstance(client: networkClient)
    }()
}
```

Swift Retrofit

- @SwiftRetrofit
- @Get
- @Query

```
@SwiftRetrofit
public protocol NewsService {

    @Get(path: "everything")
    func loadNews(@Query("q") query: String)async throws ->NewsList

}
```

Swift Retrofit

- @SwiftRetrofit

```
@SwiftRetrofit
public protocol NewsService {

    @Get(path: "everything")
    func loadNews(@Query("q") query: String)async throws ->NewsList

}
```

Swift Retrofit

@SwiftRetrofit

- Peer
- Extension

```
@attached(member, names: arbitrary)
@attached(extension)
@attached(peer, names: prefixed(SwiftRetrofit))
public macro SwiftRetrofit() =
        #externalMacro(module: "SwifRetrofitMacros",
                       type: "SwiftRetrofitMacro")
```

Swift Retrofit

@SwiftRetrofit

- Peer – доступ к узлу
- Extension – для расширения типа

```
@attached(member, names: arbitrary)
@attached(extension)
@attached(peer, names: prefixed(SwiftRetrofit))
public macro SwiftRetrofit() =
        #externalMacro(module: "SwifRetrofitMacros",
                       type: "SwiftRetrofitMacro")
```

Swift Retrofit

@Get

```
@SwiftRetrofit
public protocol NewsService {

    @Get(path: "everything")
    func loadNews(@Query("q") query: String)async throws ->NewsList

}
```

Swift Retrofit

@Get

- Peer – для доступа к узлу

```
@attached(peer, names: arbitrary)
public macro Get(path: String) =
                    #externalMacro(module: "SwifRetrofitMacros",
                                   type: "GetMacro")
```

Swift Retrofit

- @Query

```
@SwiftRetrofit
public protocol NewsService {

    @Get(path: "everything")
    func loadNews(@Query("q") query: String) async throws ->NewsList

}
```

Swift Retrofit

- @Query
- Member – просто помечаем узел (empty)

```
@attached(member)
public macro Query(name: String) =
                    #externalMacro(module: "SwifRetrofitMacros",
                                   type: "QueryMacro")
```

Swift Retrofit. План-капкан

- 1. Проверить, что декларация @SwiftRetrofit протокол
- 2. Найти все методы, помеченные @Get
- 2.1 Собрать все параметры @Query
- 4. Сгенерировать класс Retrofit по шаблону (с сетевым клиентом и плюшками)
- 5. Для каждого @Get генерируем реализацию метода по шаблону
- 5.1 Добавить обработку параметров @Query

SwiftRetrofit. Декларация = протокол

Макрос @SwiftRetrofit можно прикрепить только к протоколу

```
public struct SwiftRetrofitMacro: PeerMacro {

    public static func expansion(
        of node: SwiftSyntax.AttributeSyntax,
        providingPeersOf declaration: some DeclSyntaxProtocol,
        in context: some MacroExpansionContext
    ) throws -> [SwiftSyntax.DeclSyntax] {
        //Макрос может быть прикреплен только к протоколу!!
        guard let protocolSyntax = declaration.as(ProtocolDeclSyntax.self) else {
            return []
        }
        return DeclSyntax(newClassDecl) //<- Сейчас это напишем
    }

}
```

SwiftRetrofit. Собираем информацию

Название нашего протокола

+

Инфо по Get функциям

```swift
//Название нашего протокола
let name = protocolSyntax.name.text

//Собираем инфо по функциям
let functions = protocolSyntax.memberBlock.members.map { member in
    //Генерация для Get
    generateGetFunctionDecl(syntax: member, context: context)
}
```

SwiftRetrofit. Создание класса по шаблону

Наша цель

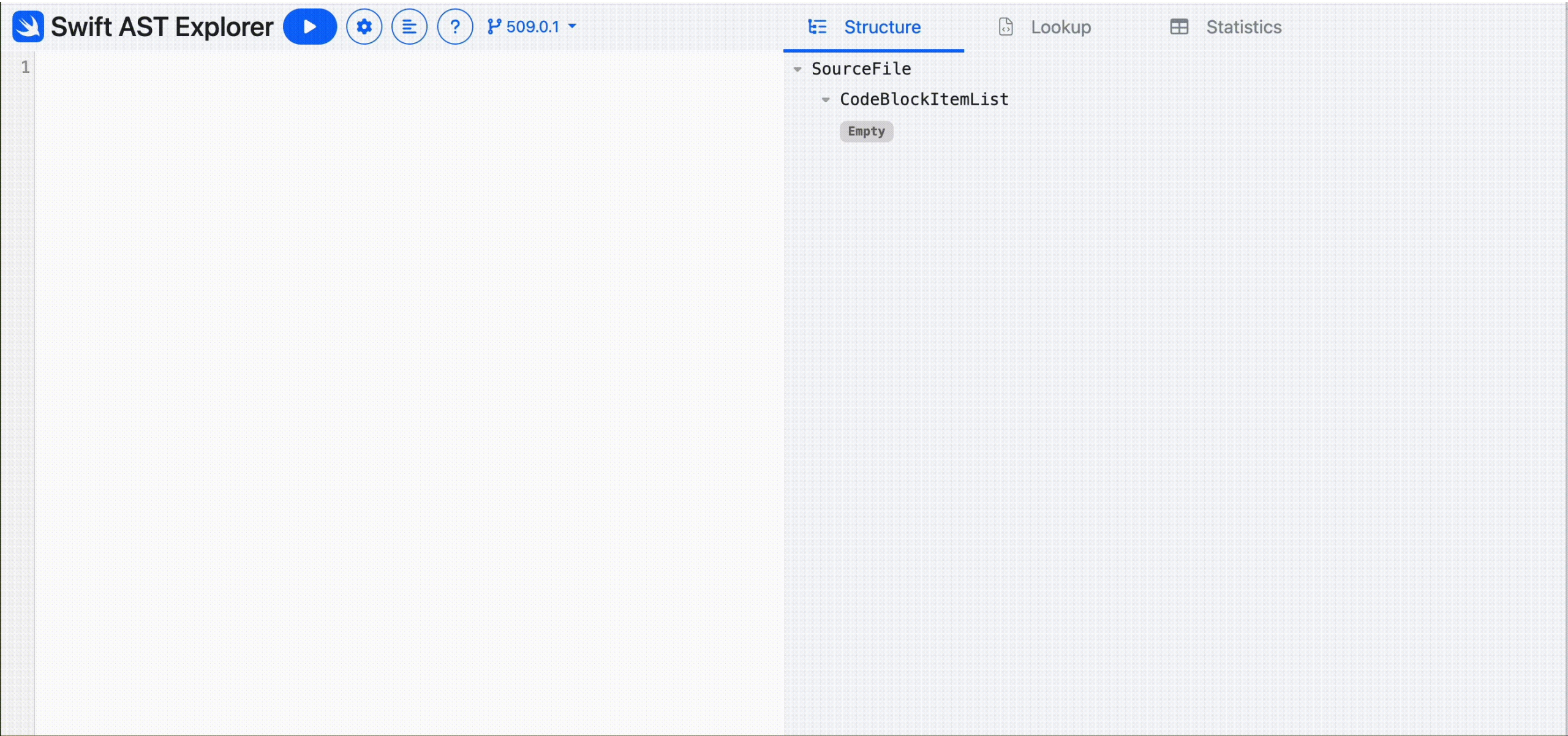
```
public class SwiftRetrofitNewsService: NewsService, NetworkServiceProtocol {

    public static func createInstance(client: NetworkClient) ->
        any NewsService {
            return SwiftRetrofitNewsService(client: client)
        }

    private let client: NetworkClient

    private init(client: NetworkClient) {
        self.client = client
    }
    //Далее Magic код
    //
    //...
}
```

AST Explorer в помощь



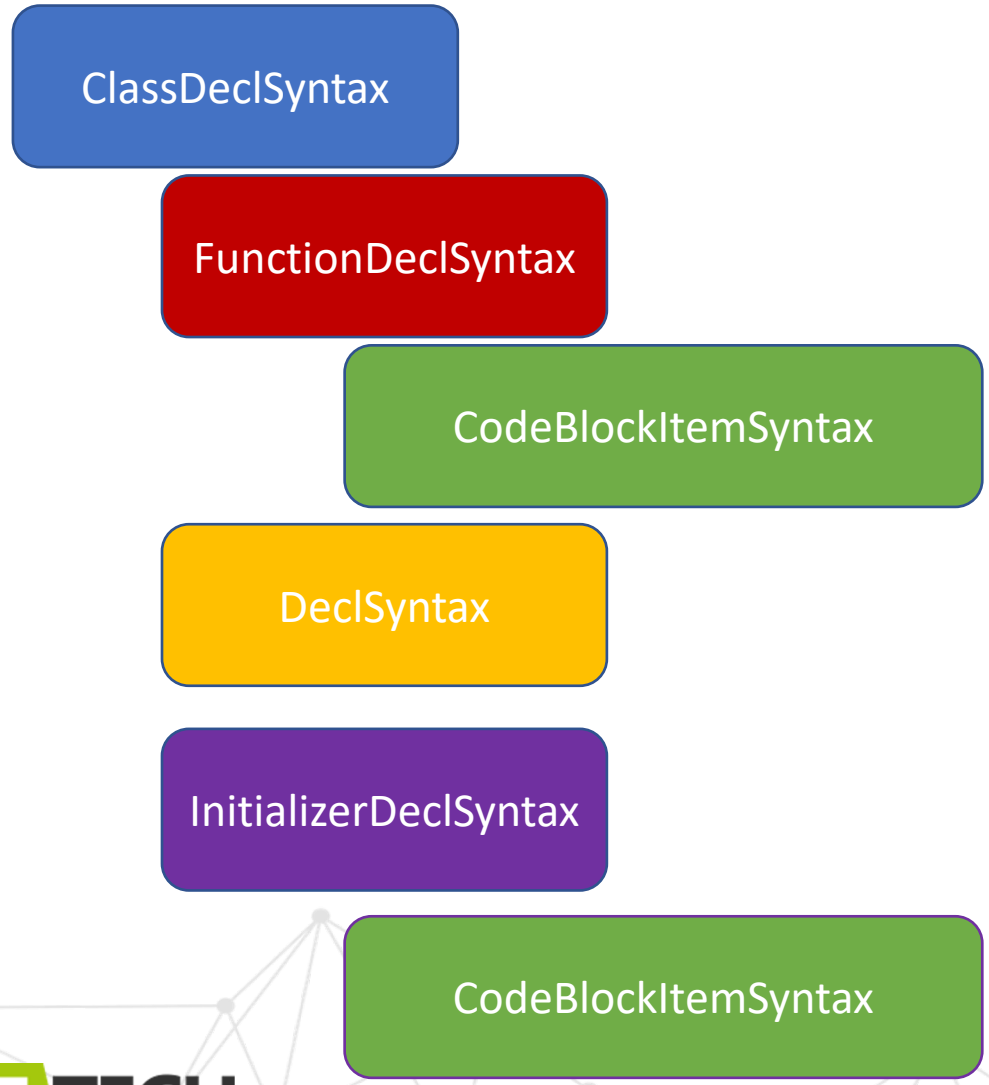
SwiftRetrofit. Создание класса по шаблону

```
let classDecl = try ClassDeclSyntax("public class
    \(raw: "SwiftRetrofit\(name)"): \(raw: name),NetworkServiceProtocol") {
        try FunctionDeclSyntax("public static func createInstance(client:
        NetworkClient) -> any \(raw: "\(name)")") {
            CodeBlockItemSyntax(item: .decl("return \(raw: "SwiftRetrofit\(name)")(client: client)"))
        }

        DeclSyntax("private let client: NetworkClient")

        try InitializerDeclSyntax("private init(client: NetworkClient)") {
            CodeBlockItemListSyntax([
                .init(item: .decl("self.client = client"))
            ])
        }
        //<-сюда вставим функции
    }
```

SwiftRetrofit. Создание класса по шаблону



SwiftRetrofit. Создание класса по шаблону

```
let classDecl = try ClassDeclSyntax("public class
    \(raw: "SwiftRetrofit\(name)"): \(raw: name),NetworkServiceProtocol") {
        try FunctionDeclSyntax("public static func createInstance(client:
        NetworkClient) -> any \(raw: "\(name)")") {
            CodeBlockItemSyntax(item: .decl("return \(raw: "SwiftRetrofit\(name)")(client: client)"))
        }

        DeclSyntax("private let client: NetworkClient")

        try InitializerDeclSyntax("private init(client: NetworkClient)") {
            CodeBlockItemListSyntax([
                .init(item: .decl("self.client = client"))
            ])
        }
        //<-сюда вставим функции
    }
```

SwiftRetrofit. Разберем функции

Исходная

Цель

```
//Протокол
@Get(path: "everything")
    func loadNews(query: QueryParam<String>)async throws ->NewsList

//Реализация Get
public func loadNews(query: QueryParam<String>) async throws -> NewsList {
  guard let url = createUrl(path: "everything",
                            queryItems:
                             .init(name: query.0, value: String(describing: query.1))])
                             else {
                        throw NetworkError.parameterError("failed to create URLComponents")
        }
         return try await client.request(urlComponents: url)
    }
```

SwiftRetrofit. Разберем функции

Исходная

Цель

```
//Протокол
@Get(path: "everything")
    func loadNews(query: QueryParam<String>)async throws ->NewsList

//Реализация Get
public func loadNews(query: QueryParam<String>) async throws -> NewsList {
  guard let url = createUrl(path: "everything",
                            queryItems:
                             .init(name: query.0, value: String(describing: query.1))])
                             else {
                        throw NetworkError.parameterError("failed to create URLComponents")
        }
         return try await client.request(urlComponents: url)
    }
```

SwiftRetrofit. Разберем функции

Swift AST Explorer 509.0.1

```
1 public func loadNews(query: QueryParam<String>) async throws -> NewsList
  {
2   guard let url = createUrl(path: "everything",
3                             queryItems: [ .init(name: query.0, value:
  String(describing: query.1))]) else {
4       throw NetworkError.parameterError("failed to create
  URLComponents")
5       }
6        return try await client.request(urlComponents: url)
7    }
```

Structure | Lookup | Statistics

- SourceFile
  - CodeBlockItemList
    - CodeBlockItem
      - FunctionDecl
        - AttributeList
        - DeclModifierList
          - DeclModifier
            - **public**
        - **func**
        - **loadNews**
        - FunctionSignature
          - FunctionParameterClause
            - **(**
            - FunctionParameterList
              - FunctionParameter
                - AttributeList
                - DeclModifierList
                - **query**
                - **:**
                - IdentifierType
                - **QueryParam**
                - GenericArgumentClause

SwiftRetrofit. Разберем функции

```
static func generateGetFunctionDecl(syntax: MemberBlockItemListSyntax.Element,
                                    context: some MacroExpansionContext)
                                    -> DeclSyntax {
    return generateFunctionDecl(syntax: syntax, context: context, attributeName: "Get") {
        funcDecl, getAttr in
        guard let appendPathExpr = getAttr.arguments?.as(LabeledExprListSyntax.self)?
        .first?.expression else {
            return ""
        }
        let returnType = funcDecl.signature.returnClause?.type.description ?? ""
        let parameters = funcDecl.signature.parameterClause.parameters
        .map { $0.as(FunctionParameterSyntax.self) }
        generateQueryItems()
        return """
        public func \(funcDecl.name)\(raw: funcDecl.signature.description) {
            guard let url = createUrl(path: \(raw: path), queryItems: [ \(raw: queryItemsStatement)])
            else {
                throw NetworkError.parameterError("failed to create URLComponents")
            }
            return try await client.request(urlComponents: url)
        }
        """
    }
}
```

SwiftRetrofit. Разберем функции

```
static func generateGetFunctionDecl(syntax: MemberBlockItemListSyntax.Element,
                                    context: some MacroExpansionContext)
                                    -> DeclSyntax {
    return generateFunctionDecl(syntax: syntax, context: context, attributeName: "Get") {
        funcDecl, getAttr in
        guard let appendPathExpr = getAttr.arguments?.as(LabeledExprListSyntax.self)?
        .first?.expression else {
            return ""
        }
        let returnType = funcDecl.signature.returnClause?.type.description ?? ""
        let parameters = funcDecl.signature.parameterClause.parameters
        .map { $0.as(FunctionParameterSyntax.self) }
        generateQueryItems()
        return """
        public func \(funcDecl.name)\(raw: funcDecl.signature.description) {
            guard let url = createUrl(path: \(raw: path), queryItems: [ \(raw: queryItemsStatement)])
            else {
                throw NetworkError.parameterError("failed to create URLComponents")
            }
            return try await client.request(urlComponents: url)
        }
        """
    }
}
```

Swift Retrofit. @Query

Query параметры

```
let parameters = funcDecl.signature.parameterClause
        .parameters.map { $0.as(FunctionParameterSyntax.self) }

        parameters.forEach { parameter in
            if let parameter {
                let parameterNameLiteral = parameter.firstName.text

                if let statement = parameter.getQueryText(), !statement.isEmpty {
                    queryItems.append(statement)
                }
            }
        }
```

Swift Retrofit. @Query

- @Query – attached(member)
- Pure retrofit style

```
@SwiftRetrofit
public protocol NewsService {

    @Get(path: "everything")
    func loadNews(@Query("q") query: String)async throws ->NewsList

}
```

Swift Retrofit. @Query

- @Query – attached(member)
- Pure retrofit style

```
@SwiftRetrofit
public protocol NewsService {

    @Get(path: "everything")
    func loadNews(@Query("q") query: String) async throws ->NewsList

}
```

- Реализуемо до Xcode 15.6 beta

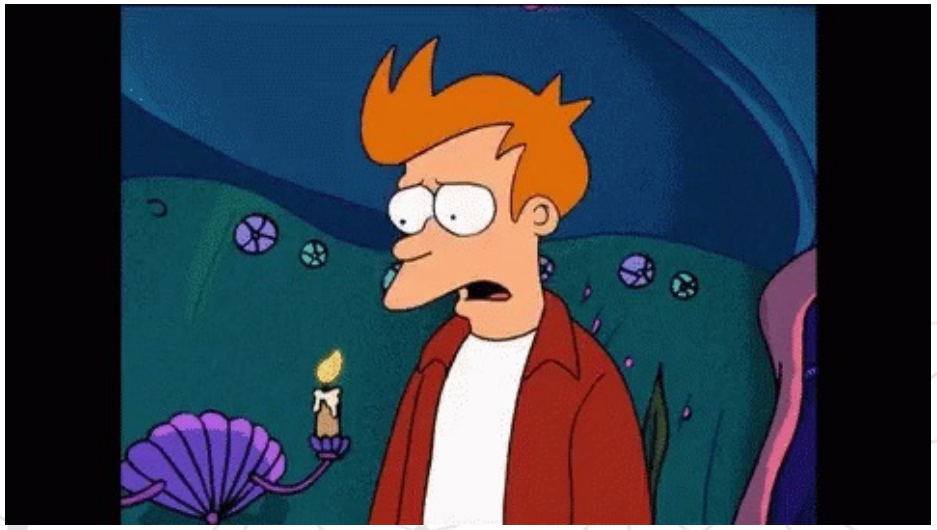
Внимание!
Теперь это работать не будет!

Изменения в финальной версии Swift 5.9

К параметрам функции макросы не применить:

```
@SwiftRetrofit
public protocol NewsService {
    @Get(path: "everything")
    func loadNews(@Query(name: "q")query:
        String)async throws ->NewsList
}
```

'member' macro cannot be attached to parameter



Изменения в финальной версии Swift 5.9

[5.9][Macros] Add missing macro validation. #67387

Merged **hborla** merged 3 commits into `apple:release/5.9` from `hborla:5.9-macro-validation` on Jul 19

Conversation 1 | Commits 3 | Checks 0 | Files changed 11

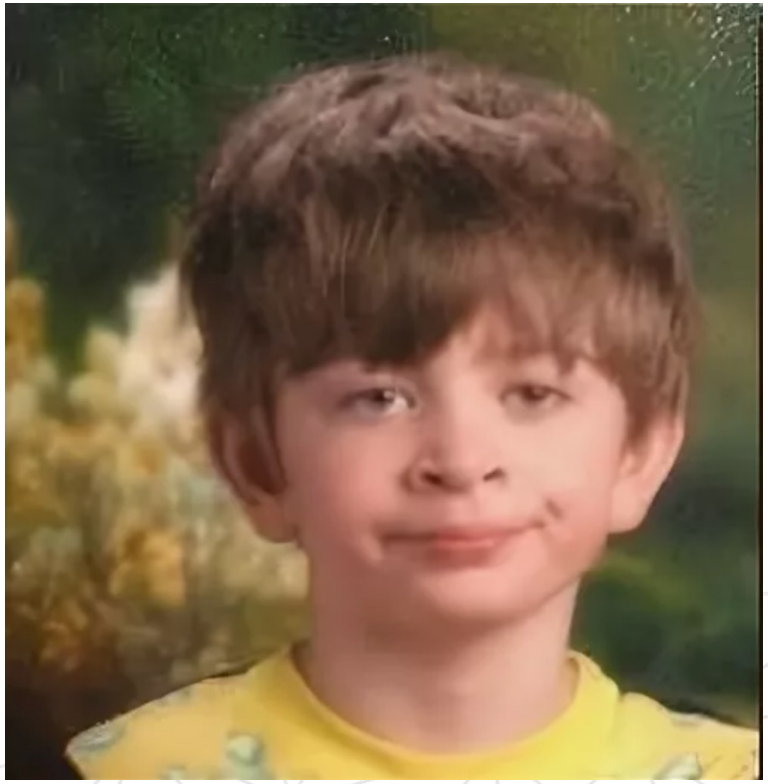
hborla commented on Jul 19 — Member

- **Explanation**: Fixes the last remaining holes (that I know of!) in macro validation, adding diagnostics for:
  - Extension macros generating names not documented in the `@attached(extension)` attribute.
  - Extension macros generating conformances not documented in the `@attached(extension)` attribute.
  - Macro attributes attached to declarations that the macro does not apply to, such as a `member` macro attached to an import declaration.
- **Scope**: Only affects attached macros.
- **Risk**: Low due to narrow scope.
- **Testing**: Added new test cases.
- **Reviewer**: @DougGregor
- **Issue**: rdar://111685434&112163965
- **Main branch PR**: [Macros] Add missing macro validation. #67106, [Macros] Diagnose undocumented conformances in extension macro expansions. #67383

Что случилось:

- «Long story short - support macros on function parameters have been removed with no immediate plans to bring them back»

  Вкратце – поддержка макросов на параметрах функций была убрана без планов на возврат в ближайшем будущем



Что делать будем

Адаптироваться – использовать псевдонимы типов

```
public typealias Path<T> = T

public typealias Header<T> = T

public typealias Query<T> = T

public typealias Field<T> = T

public typealias Body<T> = T
```

Swift Retrofit. План-капкан

- 1. Проверить, что декларация @SwiftRetroft протокол
- 2. Найти все методы, помеченные @GET
- **2.1 Собрать все параметры типа QueryParam**
- 4. Сгенерировать класс Retrofit по шаблону (с сетевым клиентом и плюшками)
- 5. Для каждого @GET генерируем реализацию метода по шаблону
- **5.1 Добавить обработку QueryParam**

Почему не PropertyWrapper

Для декларации в нужном виде в протоколах не поддерживается

```
@SwiftRetrofit
public protocol NewsService {

    @Get(path: "everything")
    func loadNewsList(@QueryPart query:
        String)async throws ->NewsList
```

parameter 'query' declared inside a protocol cannot have a wrapper

Swift Retrofit. QueryParam

Обработаем атрибуты

```
extension FunctionParameterSyntax {
    func getQueryText()->String? {
        guard let apiAttribute = self.createAttribute() else {
            return nil
        }
        return apiAttribute.apiBuilderStatement()
    }
}
```

Swift Retrofit. QueryParam

Parameter ->
ApiAttribute

```
extension FunctionParameterSyntax {

    func createAttribute()->APIAttribute? {
        let type = self.type.as(IdentifierTypeSyntax.self)?.name.text ?? ""
        let name = self.firstName.text
        switch type {
        case "QueryParam":
            return .query(key: name)
         default:
            return nil
        }
    }

}
```

Swift Retrofit. QueryParam

Parameter ->
ApiAttribute –>
Строка

Name = query.0
Key = query.1

```swift
enum APIAttribute {
    /// Атрибуты
    case query(key: String?)


    /// Генерируем строку
    func apiBuilderStatement(input: String? = nil) -> String? {
        switch self {
            case let .query(key):
            return
            ".init(name: \(key ?? "").0,
            value: String(describing:\(key ?? "").1))"

    }
}
```

Swift Retrofit. Подключаем и проверяем

Идем через DI

```
class DI {
    static let shared = DI()

    lazy var networkClient: NetworkClient = {
        return NetworkClient.Builder(config: NetworkConfiguration())
            .build()
    }()

    lazy var newsService: NewsService? = {
        return SwiftRetrofitNewsService
            .createInstance(client: networkClient)
    }()
}
```

Swift Retrofit. Подключаем и проверяем

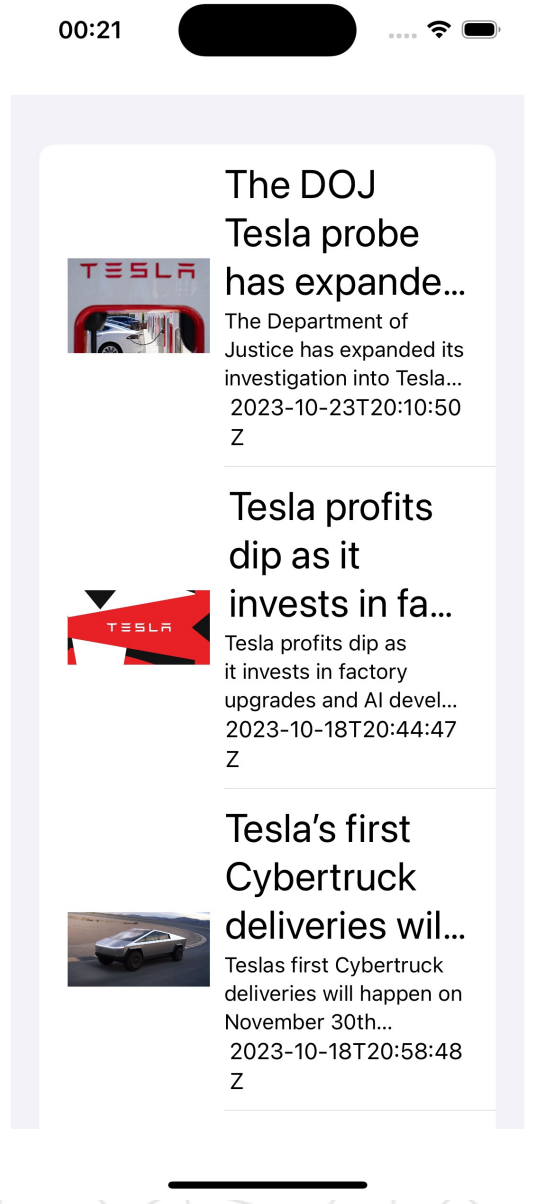
Подключаем в ViewModel

```
@Observable class NewsViewModel {
    var newsItems = [NewsItem]()
    var service = DI.shared.newsService

    @MainActor
    func load() {
        Task { [weak self] in
            guard let self = self else {return}
            do {
                let items = try await service?.loadNews(query: ("q", <query>))
                self.newsItems.removeAll()
                self.newsItems.append(contentsOf: items?.articles ?? [])
            }catch {
                print(error)
            }
        }
    }
}
```

Swift Retrofit. Подключаем и проверяем

Работает



USETECH

Теперь займемся UI
Сгенерируем!

Swift Data Struct -> View

@SwiftView

```
@SwiftView
struct TestItem {}
extension TestItem: View {                                    @SwiftView
  public var body: some View {
      VStack {
          HStack(alignment: .top, spacing: 10) {
          }
      } .frame(minWidth: 0, maxWidth: .infinity,
          alignment: .topLeading)
    }
}
```

Swift Data Struct -> View

@SwiftView

@attached(member) – для добавления body
@attached(extension) – для расширения

```
@attached(member, names: named(body))
@attached(extension, names: arbitrary)
public macro SwiftView() =
        #externalMacro(module: "SwiftViewMacros",
                       type: "SwiftViewMacro")
```

Swift Data Struct -> View

@SwiftView

~~@attached(member) – для добавления body~~
@attached(extension) – для расширения

```
//@attached(member, names: named(body))
@attached(extension, names: arbitrary)
public macro SwiftView() =
        #externalMacro(module: "SwiftViewMacros",
                       type: "SwiftViewMacro")
```

Расширение протоколом и код должны быть покрыты одной ролью!

Swift Data Struct -> View. Добавляем расширение

ExtensionDeclSyntax

```swift
public struct SwiftViewMacro : ExtensionMacro {

    public static func expansion(of node: SwiftSyntax.AttributeSyntax,
    attachedTo declaration: some SwiftSyntax.DeclGroupSyntax,
    providingExtensionsOf
    type: some SwiftSyntax.TypeSyntaxProtocol,
    conformingTo protocols: [SwiftSyntax.TypeSyntax],
    in context: some SwiftSyntaxMacros.MacroExpansionContext)
    throws -> [SwiftSyntax.ExtensionDeclSyntax] {

        if !declaration.is(StructDeclSyntax.self) {
            return []
        }
        let body  = createBody(declaration: declaration) ?? ""
        let ext: DeclSyntax =
            """
            extension \(type.trimmed): View {
            \(raw: body)
            }
            """

        return [ext.cast(ExtensionDeclSyntax.self)]

    }
```

Swift Data Struct -> View. Добавим боди

Получим данные из StructDeclSyntax

```
public struct SwiftViewMacro : ExtensionMacro {

    public static func expansion(of node: SwiftSyntax.AttributeSyntax,
    attachedTo declaration: some SwiftSyntax.DeclGroupSyntax,
    providingExtensionsOf
    type: some SwiftSyntax.TypeSyntaxProtocol,
    conformingTo protocols: [SwiftSyntax.TypeSyntax],
    in context: some SwiftSyntaxMacros.MacroExpansionContext)
    throws -> [SwiftSyntax.ExtensionDeclSyntax] {

        if !declaration.is(StructDeclSyntax.self) {
            return []
        }
        let body  = createBody(declaration: declaration) ?? ""
        let ext: DeclSyntax =
            """
            extension \(type.trimmed): View {
            \(raw: body)
            }
            """

        return [ext.cast(ExtensionDeclSyntax.self)]

    }
```

Swift Data Struct -> View. Добавим боди

Контент?

Маппинг полей!

```
private static func createBody(declaration:
SwiftSyntax.DeclGroupSyntax)-> String? {
        guard let structDel = declaration.as(StructDeclSyntax.self)
        else {
            return nil
        }
        let members = structDel.memberBlock.members
        let variableDeclarations = members
        .compactMap { $0.decl.as(VariableDeclSyntax.self) }

        var mappedData = createMappedFields(variableDeclarations)

        var body = """
          public var body: some View {
        VStack {

        """
        body += "HStack(alignment: .top, spacing: 10) {\n"
        body += processMapped(mappedData)
        body += "}"

        body += """
                }.frame(minWidth: 0, maxWidth: .infinity,
                alignment: .topLeading)
            }
        """

        return body
    }
```

Swift Data Struct -> View. Добавим боди

Контент?

Маппинг полей!

```
@SwiftView
public struct Test {
    let id = UUID().uuidString
    @MappedImage
    public var name: String = ""

    @MappedText(style: .title)
    public var title: String = "Title"

    @MappedText(style: .detail)
    public var detail: String = "Detail"

    @MappedText(style: .callout)
    public var date: String = "Date"
}
```

Swift Data Struct -> View. Маппинг полей

@MappedImage

@MappedText

@attached(peer) – доступ к узлу

```
@attached(peer)
public macro MappedImage(_ value: String = "")
            = #externalMacro(module: "SwiftViewMacros",
                             type: "MappedField")

@attached(peer)
public macro MappedText(style : TextStyle)
            = #externalMacro(module: "SwiftViewMacros",
                             type: "MappedField")
```

Swift Data Struct -> View. Добавим боди

Маппинг деклараций

VariableDeclSyntax

```
var mappedData = [MemberData]()
    variableDeclarations.forEach { declaration in

        let name = declaration.bindings.first?.pattern ?? ""

        let type = declaration.bindings.first?
        .typeAnnotation?.type.as(IdentifierTypeSyntax.self)!
        .name.text ?? ""

        let attributeData = declaration.attributeData()

        mappedData
            .append(
                MemberData(name: "\(name)",
                typeName: type,
                attributeData: attributeData))
    }
```

Swift Data Struct -> View. Маппинг полей

VariableDeclSyntax -> AttributeData

```
extension VariableDeclSyntax {
    func attributeData()-> AttributeData {
        var attributeData: AttributeData? = nil
        var type: AttributeType = .unknown
        var name: String = ""
        var params: FieldAttributes? = nil
        self.attributes.forEach {
            item in
            if let attribute = item.as(AttributeSyntax.self) {
                name =  attribute.attributeName
                .as(IdentifierTypeSyntax.self)?.name.text ?? ""
               /// <- To AttributeData
               /// Создаем тип и маппим параметры
            }
            attributeData = AttributeData(name: name,
            type: type,
            params: params)
        }
        return attributeData ?? AttributeData(name: "", type: .unknown)
    }
}
```

Swift Data Struct -> View. Маппинг полей

VariableDeclSyntax -> AttributeData

Параметры из аргументов макроса

```
if let attribute = item.as(AttributeSyntax.self) {
    name =  attribute.attributeName
    .as(IdentifierTypeSyntax.self)?.name.text ?? ""
    if name == "MappedImage" {
        type = AttributeType.image
    }

    if name == "MappedText" {
        type = AttributeType.text
        if let style = attribute.arguments?.argExpSyntax() {
            params = TextAttributes(textStyle: "\(style
            .trimmed.description)".style(),
            name: "\(style.trimmed.description)")
        }else {
            params = TextAttributes(textStyle: .unknown)
        }
    }
}
```

Swift Data Struct -> View. Маппинг полей

VariableDeclSyntax -> AttributeData

Параметры из аргументов макроса

```
extension AttributeSyntax.Arguments {
    func argExpSyntax() -> ExprSyntax? {
        return self.as(LabeledExprListSyntax.self)!.first!.expression
    }
}
```

Swift Data Struct -> View. Добавим боди

Своя разметка по своему вкусу

Элемент – в вью по шаблону

```
var body = """
        public var body: some View {
        VStack {

        """
        let textGroup = mappedData.filter{$0.attributeData.type == .text}
        let icon = mappedData.filter{$0.attributeData.type == .image}
        body += "HStack(alignment: .top, spacing: 10) {\n"
        if !icon.isEmpty {
            body += icon.first?.toView() ?? ""
        }
        if !textGroup.isEmpty {
            body += "\nVStack(alignment: .leading) {\n"
            textGroup.forEach { text in
                body += text.toView()
            }
            body += "}"
        }
        body += "}"


        body += """
                }.frame(minWidth: 0, maxWidth: .infinity,
                alignment: .topLeading)
            }
        """
```

USETECH

Swift Data Struct -> View. Добавим боди

Своя разметка по своему вкусу

Элемент – в вью по шаблону

```
func toView(_ nodeName: String = "")->String {
        let node = nodeName.isEmpty ? "" : "\(nodeName)."
        switch (self.attributeData.type) {
        case .image:
            return toImage(node)
        case .text:
            return toText(node)
        default:
            return ""
        }

    }
```

Swift Data Struct -> View. Добавим боди

Своя разметка по своему вкусу

Элемент – в вью по шаблону

```
func toText(_ nodeName: String = "")->String {
        var body = "Text(\(nodeName)\(self.name))"
        if  let attrs = (self.attributeData.params as? TextAttributes)  {
            body += ".font(.\(attrs.textStyle.styleName)).lineLimit(3)"
        }
        return body
    }
```

Swift Data Struct -> View. Добавим боди

Своя разметка по своему вкусу

Элемент – в вью по шаблону

```
func toImage(_ nodeName: String = "")->String {
        return """
        let image = \(nodeName)\(self.name) ??  \"\"
        if !image.isEmpty {
            AsyncImage(url: URL(string: image)){ image in
                image
                    .resizable()
                    .aspectRatio(contentMode: .fit)
            } placeholder: {
                Color.gray
            }
            .frame(width: 100, height: 100)
         } else {
            Color.gray.frame(width: 100, height: 100)
        }
        """
    }
```

Swift Data Struct -> View. Результат

И отдельный вью, и заготовка

Переделаем в элемент списка и соберем экран

SwiftUI ListItem. Было

@SwiftView

И View, и модель

```
@SwiftView
public struct NewsItem: Codable {
    /*...*/
}

extension NewsItem : View {
    var body : some View {
        VStack {
            /** контент */
        }
    }
}
```

SwiftUI ListItem. Хотим

@ListItem

View отдельно,
модель - отдельно

```
@ListItem
public struct NewsItem: Codable {
    /*...*/
}

public NewsItemView : View {
    var body : some View {
        VStack {
            /** контент */
        }
    }
}
```

SwiftUI ListItem. @ListItem

@ListItem

```
@ListItem
public struct NewsItem: Codable {

    let id = UUID().uuidString
    @MappedImage
    var urlToImage: String = ""
    @MappedText(style: .title)
        var title: String = ""

    @MappedText(style: .detail)
        var content: String = ""
    @MappedText(style: .callout)
    var publishedAt: String = ""
}
```

SwiftUI ListItem. @ListItem

@ListItem

Peer – генерируем новую структуру NewsItemRow

Extension – вспомогательное расширение модели

Member – вспомогательное (заглушка)

```
@attached(member, names: arbitrary)
@attached(peer, names: suffixed(ItemRow))
@attached(extension, names: arbitrary)
public macro ListItem() =
        #externalMacro(module: "SwiftViewMacros",
                       type: "ListItemMacro")
```

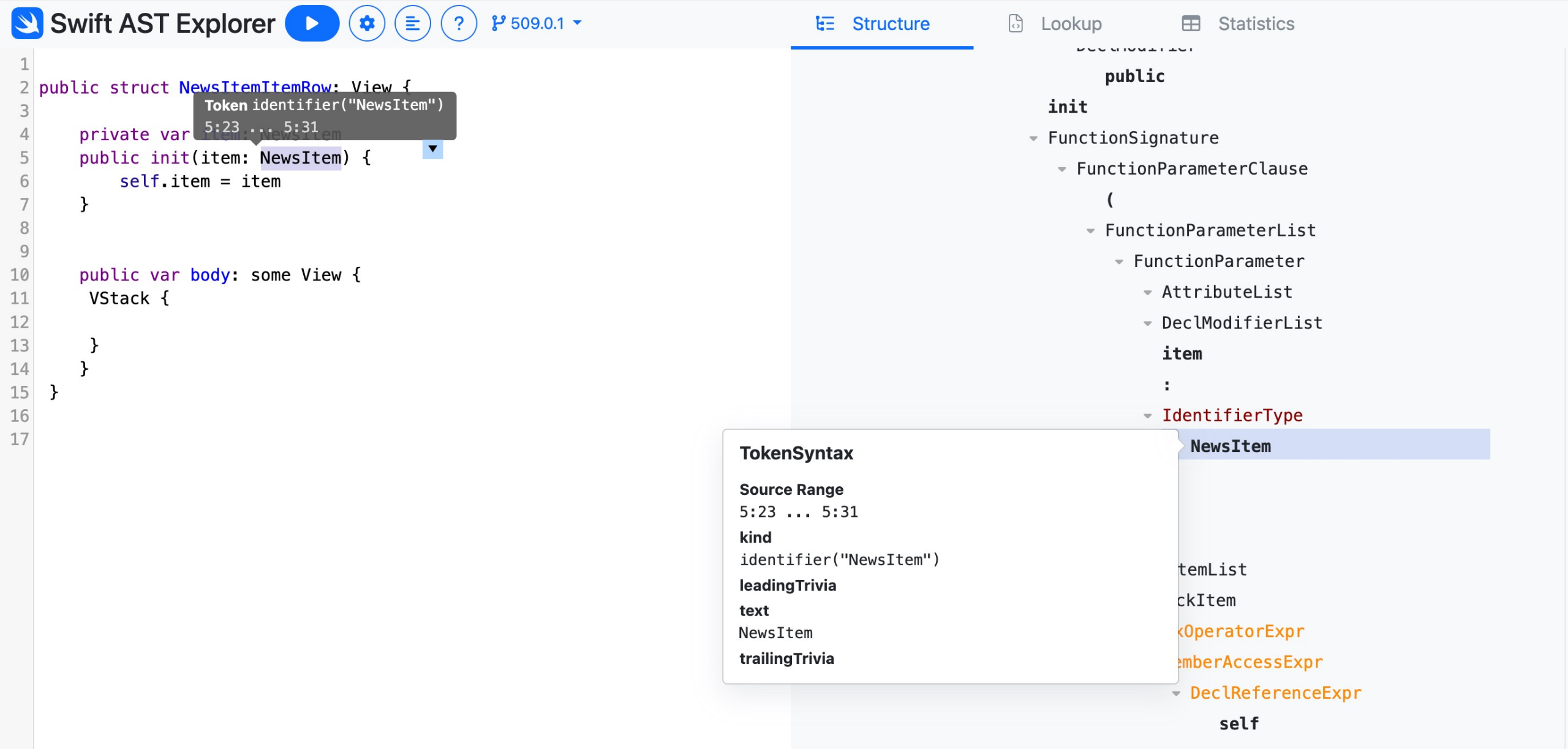
SwiftUI ListItem. @ListItem

Новая структура

```
###unwrap
public struct NewsItemItemRow: View {
    private var item: NewsItem
    public init(item: NewsItem) {
        self.item = item
    }


    public var body: some View {
     VStack {
         /// body mapping
     }
    }
}
```

SwiftUI ListItem. @ListItem. AST



SwiftUI ListItem. @ListItem

Новая структура

```
public static func expansion(of node: SwiftSyntax.AttributeSyntax,
    providingPeersOf declaration: some SwiftSyntax.DeclSyntaxProtocol,
    in context: some SwiftSyntaxMacros.MacroExpansionContext)
    throws -> [SwiftSyntax.DeclSyntax] {
    guard let structDeclSyntax = declaration
    .as(StructDeclSyntax.self) else {
        return []
    }

    let name = structDeclSyntax.name.text
    let members = structDeclSyntax.memberBlock.members

    var mappedData = createMappedData(members)
    let body = preparyBody(mappedData)

    let classDecl = try StructDeclSyntax("public struct \(raw: "\(name)ItemRow"): View") {
        DeclSyntax("private var item: \(raw: name)")
        try InitializerDeclSyntax("public init(item: \(raw: name))") {
            CodeBlockItemListSyntax([
                .init(item: .decl("self.item = item"))

            ])
        }
      createBody(body) ///<- Знакомый маппинг
    }

    return [DeclSyntax(classDecl)]
}
```

SwiftUI ListItem. @ListItem

Маппинг мы уже умеем

```
public static func expansion(of node: SwiftSyntax.AttributeSyntax,
    providingPeersOf declaration: some SwiftSyntax.DeclSyntaxProtocol,
    in context: some SwiftSyntaxMacros.MacroExpansionContext)
    throws -> [SwiftSyntax.DeclSyntax] {
    guard let structDeclSyntax = declaration
    .as(StructDeclSyntax.self) else {
        return []
    }

    let name = structDeclSyntax.name.text
    let members = structDeclSyntax.memberBlock.members

    var mappedData = createMappedData(members)
    let body = preparyBody(mappedData)

    let classDecl = try StructDeclSyntax("public struct \(raw: "\(name)ItemRow"): View") {
        DeclSyntax("private var item: \(raw: name)")
        try InitializerDeclSyntax("public init(item: \(raw: name))") {
            CodeBlockItemListSyntax([
                .init(item: .decl("self.item = item"))

            ])
        }
     createBody(body) ///<- Знакомый маппинг
    }

    return [DeclSyntax(classDecl)]
}
```

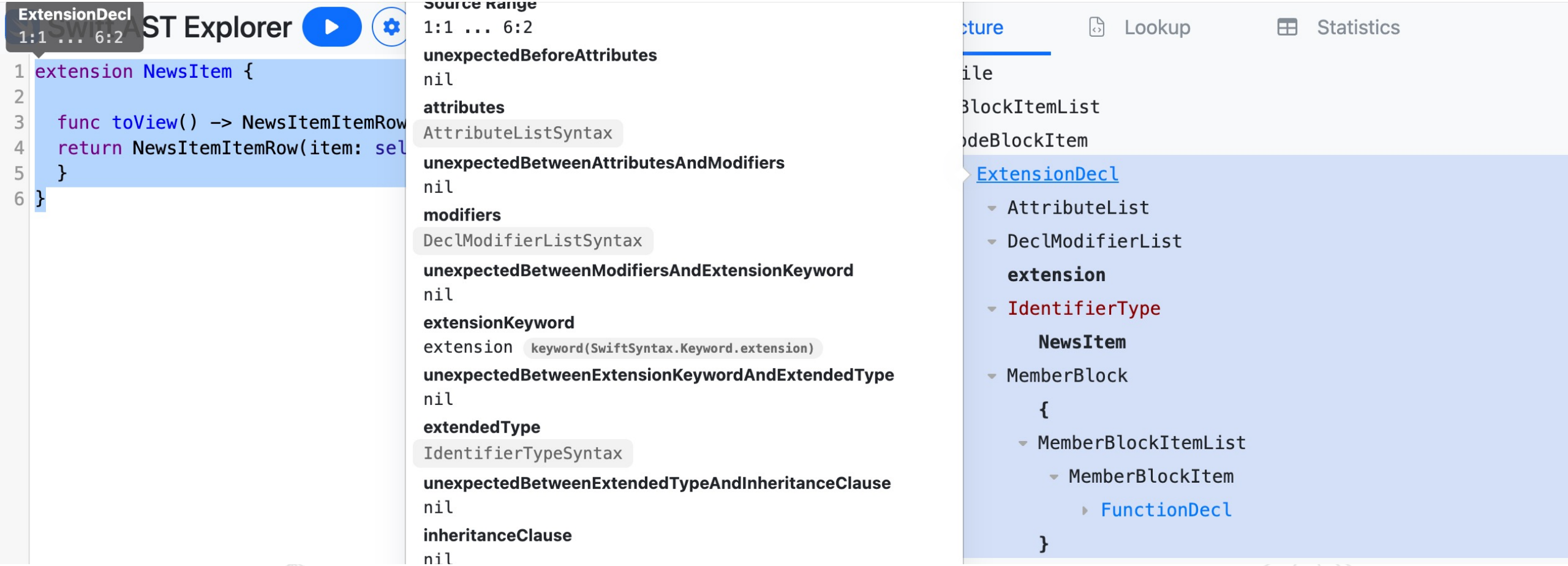
SwiftUI ListItem. NewsItem -> NewsItemRow

Модель –> View

```
extension NewsItem {                                    @ListItem

  func toView() -> NewsItemItemRow {
  return NewsItemItemRow(item: self)
  }
}
```

SwiftUI ListItem. NewsItem -> NewsItemRow



SwiftUI ListItem. NewsItem -> NewsItemRow

ExtensionDeclSyntax

```
public static func expansion(of node: SwiftSyntax.AttributeSyntax,
                             attachedTo declaration: some SwiftSyntax.DeclGroupSyntax,
                             providingExtensionsOf type: some SwiftSyntax.TypeSyntaxProtocol,
                             conformingTo protocols: [SwiftSyntax.TypeSyntax],
                             in context: some SwiftSyntaxMacros.MacroExpansionContext)
                             throws -> [SwiftSyntax.ExtensionDeclSyntax] {

        if !declaration.is(StructDeclSyntax.self) {
            return []
        }
        let ext: DeclSyntax =
            """
            extension \(type.trimmed) {

              func toView()-> \(type.trimmed)ItemRow {
              return \(type.trimmed)ItemRow(item: self)
              }
            }
            """

        return [ext.cast(ExtensionDeclSyntax.self)]
    }
```

Дело за списком

SwiftUI ListScreen. Экран списка

@ListScreen

@ListItemData

```
@ListScreen
public struct NewsList: Codable {
    var status: String?
    var total: Int = 0

    @ListItemData
    var articles: [NewsItem]

}
```

SwiftUI ListScreen. @ListScreen

@ListScreen

Peer – создаем новую структуру

```
@attached(peer, names: suffixed(ItemsScreen))
public macro ListScreen()
            = #externalMacro(module: "SwiftViewMacros",
                             type: "ListMacro")
```

SwiftUI ListScreen. @ListItemData

@ListItemData

Peer - маркер

```
@attached(peer, names: arbitrary)
public macro ListItemData()
                = #externalMacro(module: "SwiftViewMacros",
                                 type: "ListItemDataMacro")
```

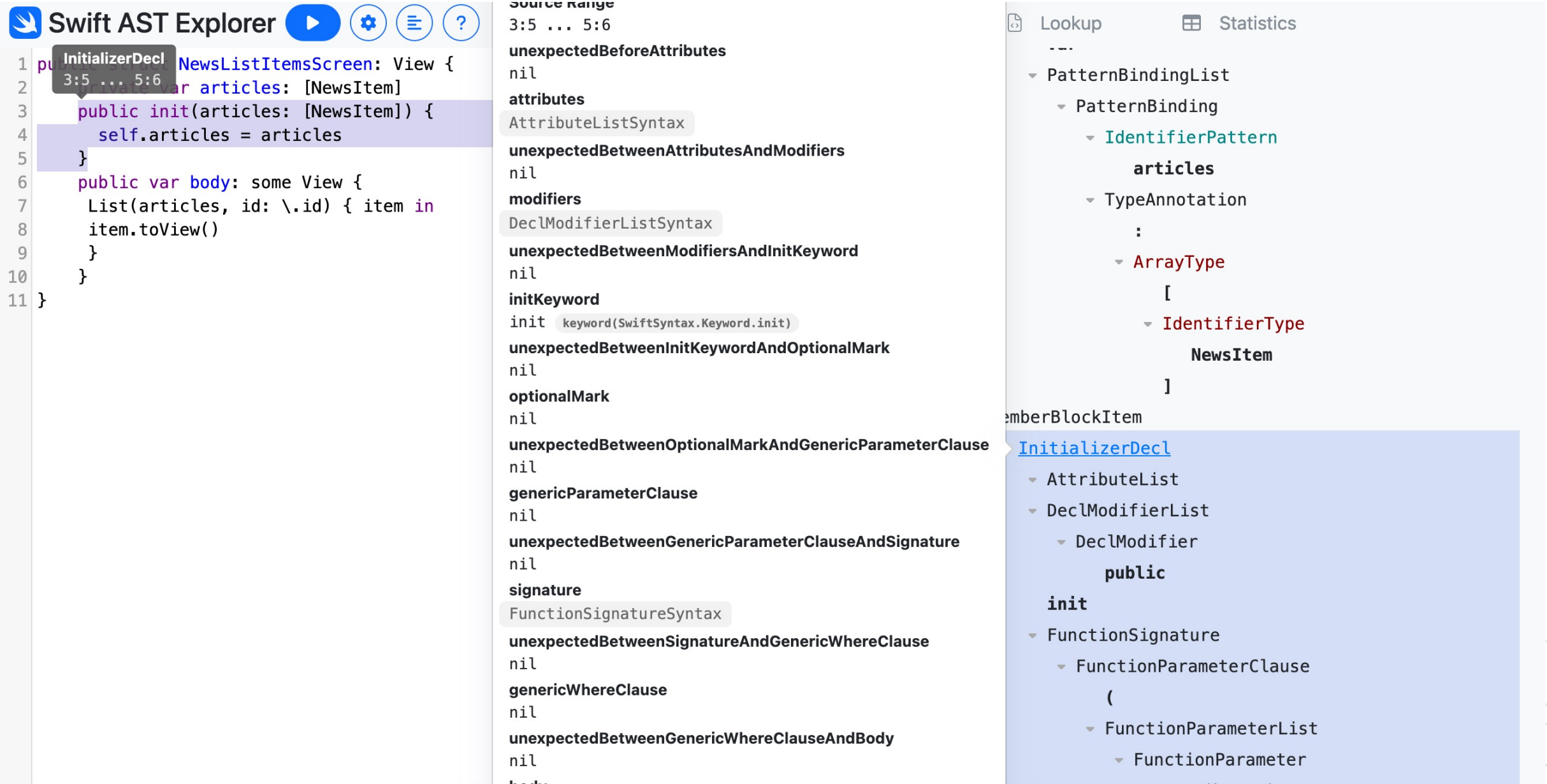
SwiftUI ListScreen. Что хотим

@ListScreen

@ListItemData

```
1  public struct NewsListItemsScreen: View {    @ListScreen
2      private var articles: [NewsItem]
3      public init(articles: [NewsItem]) {self.articles
           = articles
4      }
5      public var body: some View {
6       List(articles, id: \.id) { item in
7       item.toView()
8       }
9      }
10 }
```

SwiftUI ListScreen. AST



SwiftUI ListScreen. Новая структура ItemsScreen

StructDeclSyntax

```
let classDecl = try StructDeclSyntax("public struct \(raw: "\(name)ItemsScreen"): View") {
    DeclSyntax("private var \(raw: itemsName):[\(raw: itemsType)]")
    try InitializerDeclSyntax("public init(\(raw: itemsName): [\(raw: itemsType)])") {
        CodeBlockItemListSyntax([
            .init(item: .decl("self.\(raw: itemsName) = \(raw: itemsName)"))
        ])
    }
    try VariableDeclSyntax(
        """
            public var body: some View {
                List(\(raw: itemsName),id: \\.id) { item in
                    item.toView()
                }
            }
        """)
}
```

SwiftUI ListScreen. @ListItemData

Ищем свойство, помеченное атрибутом @ListItemData

```
let itemsProperty = structDeclSyntax.memberBlock
.members.compactMap { member in
    if (member.decl.as(VariableDeclSyntax.self)?.attributes.first { e in
        e.as(AttributeSyntax.self)?.attributeName
        .as(IdentifierTypeSyntax.self)?.name.text == "ListItemData"

    }?.as(AttributeSyntax.self) != nil)
    {
        (member.decl.as(VariableDeclSyntax.self))
        } else {
            nil
        }
    }

let itemsName = itemsProperty.first?.bindings.first?.pattern ?? ""

let itemsType = itemsProperty.first?.bindings.first?.typeAnnotation?.type
    .as(ArrayTypeSyntax.self)?.element
    .as(IdentifierTypeSyntax.self)?.name.text ?? ""
```

SwiftUI ListScreen. Подключаем

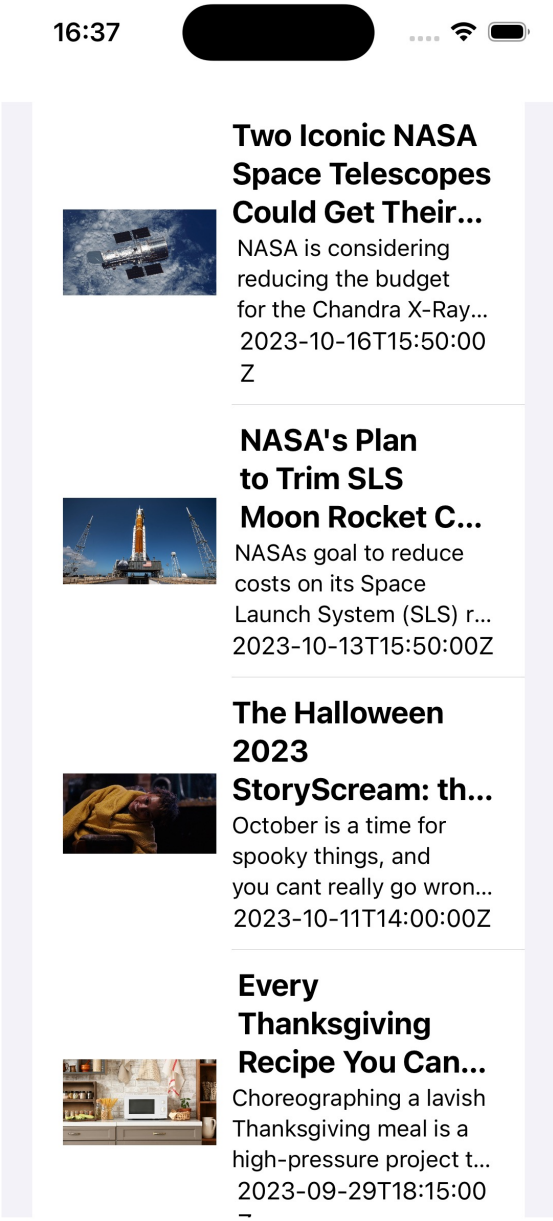
Подключаем и проверяем

```
struct ContentView: View {
    @State var model = NewsViewModel()

    var body: some View {
        VStack {
            NewsListItemsScreen(articles: model.newsItems)
            /* List(model.newsItems, id: \.id) { item in
                NewsItemRow(item: item)
            }*/
        }.onAppear {
            model.load()
        }
        .padding()
    }
}
```

SwiftUI ListScreen. Проверяем

Подключаем и проверяем



Summary

- Макросы – инновационный инструмент для генерации кода при компиляции
- Широкий выбор применения, роли можно комбинировать
- API не стабильно -> Apple меняет без документации
- Часть возможностей – забытые люки
- Инструмент развивается

Sources

https://github.com/anioutkazharkova/swiftui_complex_macro

https://developer.apple.com/documentation/Swift/applying-macros

https://developer.apple.com/documentation/swift/macros

https://swift-ast-explorer.com/

https://swift.org/blog/swift-5.9-released/

https://www.avanderlee.com/swift/macros/

https://developer.apple.com/videos/play/wwdc2023/10166/

https://developer.apple.com/videos/play/wwdc2023/10167

Спасибо за внимание!

@anioutkajarkova

azharkova
prettygeeknotes