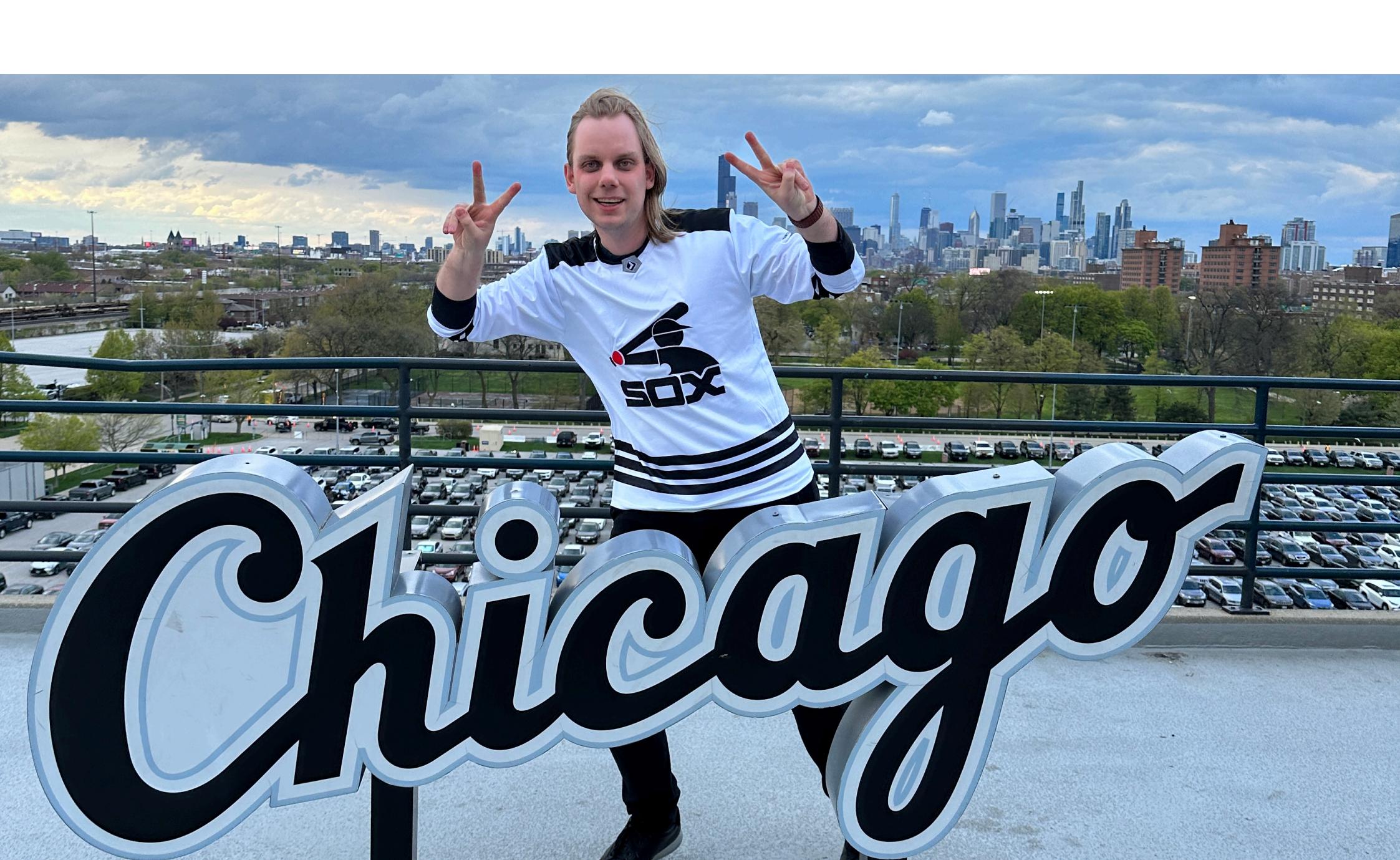
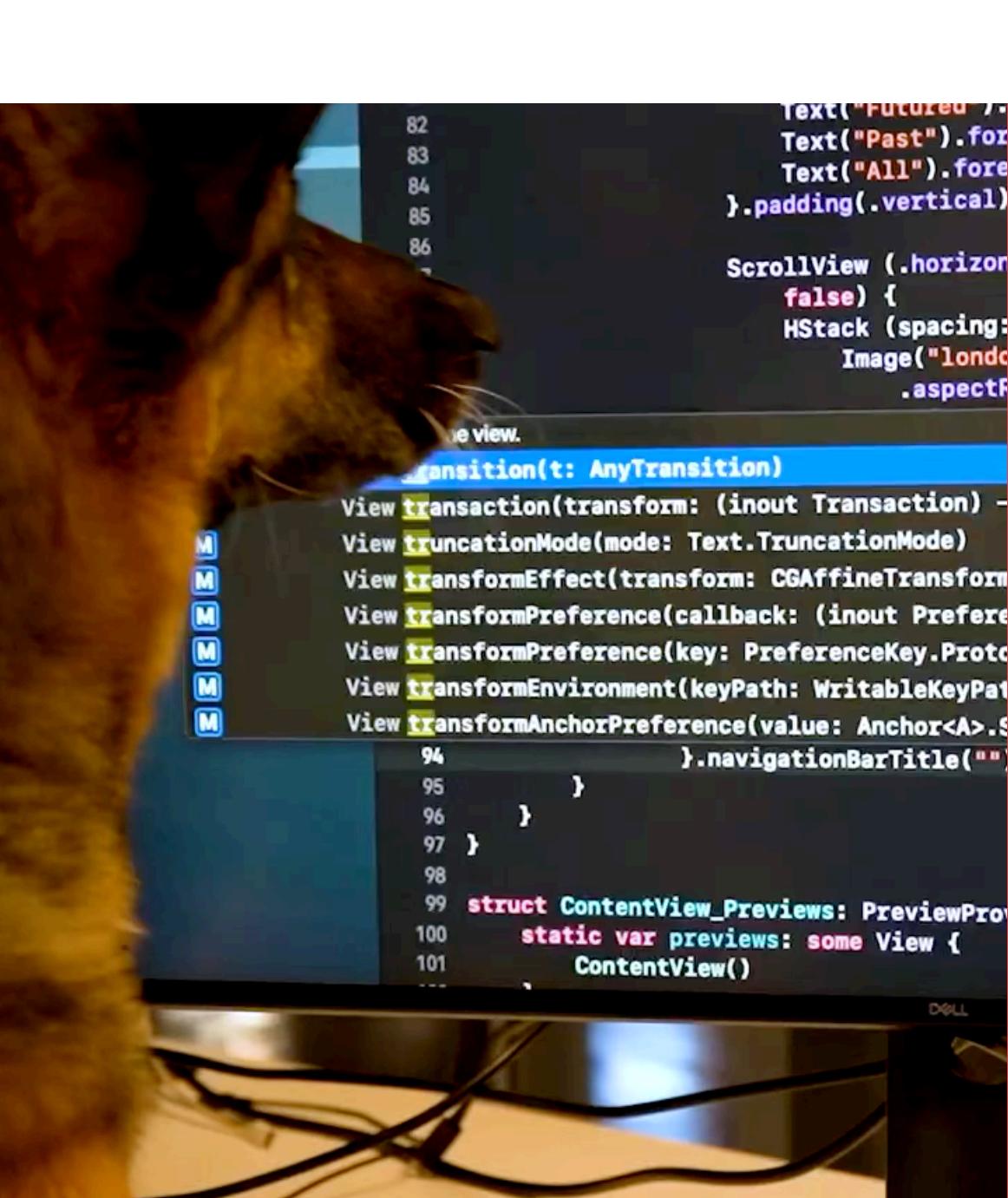


Documenting Your Project With DocC



SIMONBS.DEV





```
82     Text("Future").font(.bold)
83     Text("Past").font(.bold)
84     Text("All").font(.bold)
85   }.padding(.vertical)
86
87   ScrollView (.horizontalScrolling(false)) {
88     HStack (spacing: 10) {
89       Image("london")
90     }.aspectRatio(1).edgesIgnoringSafeArea(.top)
91   }
92
93   .navigationBarTitle("Content View")
94
95 }
96
97 }
98
99 struct ContentView_Previews: PreviewProvider {
100   static var previews: some View {
101     ContentView()
```





Runestone



Scriptable



Jayson



Re: Toot



Data Jar



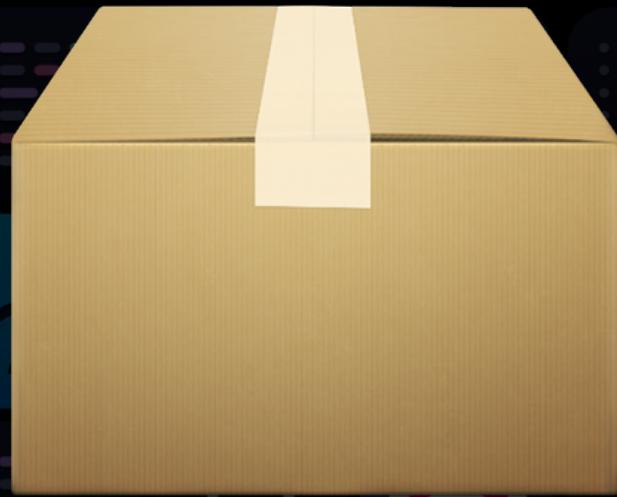
Sticky Draw



Runestone

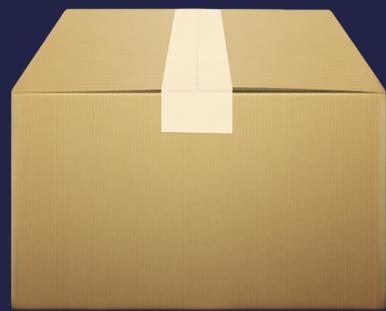


Runestone

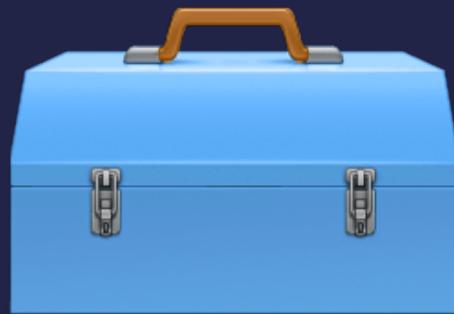


Runestone





Swift Packages



Frameworks



Apps

How can I document my project?



The screenshot shows a web browser window displaying the Apple Developer documentation for the `SCNAnimation` class. The URL in the address bar is `developer.apple.com`. The page title is "SCNAnimation". The top navigation bar includes links for News, Discover, Design, Develop, Distribute, Support, Account, and a search icon. A sidebar on the left is titled "SceneKit" and contains a tree view of SceneKit components, including First Steps, SCNScene, SCNView, Scene Structure, SCNNODE, SCNReferenceNode, Display and Interactivity, SCNSceneRenderer, SCNSceneRendererDelegate, SCNLayer (Deprecated), SCNRenderer, SCNHitTestResult, Lighting, Cameras, and Shading, SCNLight, SCNCamera, SCNMaterial, SCNMaterialProperty, Geometry, SCNGeometry, SCNGeometrySource, SCNGeometryElement, Built-in Geometry Types, Animation and Constraints, Animation, Constraints, SCNSkinner, SCNMorpher, Physics, Physics Simulation, Particle Systems, SCNParticleSystem, and SCNParticlePropertyController. A "Filter" button is at the bottom of the sidebar.

Class

SCNAnimation

(iOS 11.0+) (iPadOS 11.0+) (macOS 10.13+) (Mac Catalyst 13.1+) (tvOS 11.0+) (watchOS 4.0+)

Declaration

```
class SCNAnimation : NSObject
```

Topics

Supporting Types

```
typealias SCNAnimationDidStartBlock  
typealias SCNAnimationDidStopBlock
```

Initializers

```
init(caAnimation: CAAnimation)  
init(contentsOf: URL)  
init(named: String)
```

Instance Properties

```
var animationDidStart: SCNAnimationDidStartBlock?  
var animationDidStop: SCNAnimationDidStopBlock?  
var animationEvents: [SCNAnimationEvent]?  
var autoreverses: Bool  
var blendInDuration: TimeInterval  
var blendOutDuration: TimeInterval  
var duration: TimeInterval
```

The screenshot shows the Apple Developer documentation for the `SCNAnimation` class. The sidebar on the left lists various SceneKit components, many of which are marked as "Not documented" with pink arrows pointing to them from the main content area. The main content area includes sections for Declaration, Topics, Supporting Types, Initializers, Instance Properties, and a code snippet for the `SCNAnimation` class.

Not documented

Class

SCNAnimation

(iOS 11.0+) (iPadOS 11.0+) (macOS 10.13+) Mac Catalyst 13.1+ tvOS 11.0+ watchOS 4.0+

Declaration

```
class SCNAnimation : NSObject
```

Topics

Supporting Types

```
typealias SCNAnimationDidStartBlock
typealias SCNAnimationDidStopBlock
```

Initializers

```
init(caAnimation: CAAnimation)
init(contentsOf: URL)
init(named: String)
```

Instance Properties

```
var animationDidStart: SCNAnimationDidStartBlock?
var animationDidStop: SCNAnimationDidStopBlock?
var animationEvents: [SCNAnimationEvent]?
var autoreverses: Bool
var blendInDuration: TimeInterval
var blendOutDuration: TimeInterval
var duration: TimeInterval
```



developer.apple.com

Apple Developer News Discover Design Develop Distribute Support Account

Documentation / WidgetKit

Language: Swift API Changes: Show ▾

WidgetKit

Framework

WidgetKit

Show relevant, glanceable content from your app as widgets in iOS and macOS, and as watch complications.

iOS 14.0+ iPadOS 14.0+ macOS 11.0+ Mac Catalyst 14.0+



Overview

WidgetKit gives users ready access to content in your app by putting widgets on the iOS Home Screen and Today View, the macOS Notification Center, and by putting accessory widgets on the Lock Screen on iPhone and as complications in watchOS. Your widgets stay up to date so users always have the latest information at a glance. When they need more details, your widget takes them directly to the appropriate place in your app.

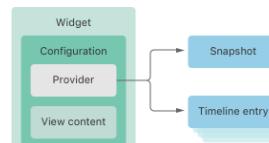
With different sizes (small, medium, large, extra large) and accessory styles (circular, rectangular, flat, and corner in watchOS), widgets can display a wide range of information. Users can personalize widgets to see details specific to their needs, and arrange their widgets in whatever way works best for them. When users stack widgets on the Home Screen and enable Smart Rotate, WidgetKit automatically rotates the most relevant widget to the top, making sure users see the most important details at exactly the right time.

Starting with iOS 16 and watchOS 9, WidgetKit allows you to create accessory widgets that appear as complications in watchOS and as widgets on the Lock Screen on iPhone. Accessory widgets are a great opportunity to bring your iOS app content to Apple Watch and your watchOS app content to iPhone.

Note

You use WidgetKit to add support for Live Activities to your app. However, Live Activities aren't widgets, so you update their content differently. To learn more about Live Activities, see [ActivityKit](#).

To implement a widget, add a widget extension to your app. Configure the widget with a timeline provider, and use [SwiftUI](#) views to display the widget's content. The timeline provider tells WidgetKit when to update your widget's content.



Documentation / WidgetKit

Language: Swift API Changes: Show ▾

WidgetKit

Show relevant, glanceable content from your app as widgets in iOS and macOS, and as watch complications.

iOS 14.0+ iPadOS 14.0+ macOS 11.0+ Mac Catalyst 14.0+

Framework

Overview

WidgetKit gives users ready access to content in your app by putting widgets on the iOS Home Screen and Today View, the macOS Notification Center, and by putting accessory widgets on the Lock Screen on iPhone and as complications in watchOS. Your widgets stay up to date so users always have the latest information at a glance. When they need more details, your widget takes them directly to the appropriate place in your app.

With different sizes (small, medium, large, extra large) and accessory styles (circular, rectangular, flat, and corner in watchOS), widgets can display a wide range of information. Users can personalize widgets to see details specific to their needs, and arrange their widgets in whatever way works best for them. When users stack widgets on the Home Screen and enable Smart Rotate, WidgetKit automatically rotates the most relevant widget to the top, making sure users see the most important details at exactly the right time.

Starting with iOS 16 and watchOS 9, WidgetKit allows you to create accessory widgets that appear as complications in watchOS and as widgets on the Lock Screen on iPhone. Accessory widgets are a great opportunity to bring your iOS app content to Apple Watch and your watchOS app content to iPhone.

Note

You use WidgetKit to add support for Live Activities to your app. However, Live Activities aren't widgets, so you update their content differently. To learn more about Live Activities, see [ActivityKit](#).

To implement a widget, add a widget extension to your app. Configure the widget with a timeline provider, and use [SwiftUI](#) views to display the widget's content. The timeline provider tells WidgetKit when to update your widget's content.

```

graph TD
    Widget[Widget] --> Configuration[Configuration]
    Widget --> Provider[Provider]
    Widget --> ViewContent[View content]
    Provider --> Snapshot[Snapshot]
    ViewContent --> TimelineEntry[Timeline entry]
  
```

developer.apple.com

Apple Developer News Discover Design Develop Distribute Support Account

Language: Swift API Changes: Show ▾

Documentation / WidgetKit

WidgetKit

Framework

WidgetKit

Show relevant, glanceable content from your app as widgets in iOS and macOS, and as watch complications.

iOS 14.0+ iPadOS 14.0+ macOS 11.0+ Mac Catalyst 14.0+

Overview

WidgetKit gives users ready access to content in your app by putting widgets on the iOS Home Screen and Today View, the macOS Notification Center, and by putting accessory widgets on the Lock Screen on iPhone and as complications in watchOS. Your widgets stay up to date so users always have the latest information at a glance. When they need more details, your widget takes them directly to the appropriate place in your app.

With different sizes (small, medium, large, extra large) and accessory styles (circular, rectangular, flat, and corner in watchOS), widgets can display a wide range of information. Users can personalize widgets to see details specific to their needs, and arrange their widgets in whatever way works best for them. When users stack widgets on the Home Screen and enable Smart Rotate, WidgetKit automatically rotates the most relevant widget to the top, making sure users see the most important details at exactly the right time.

Starting with iOS 16 and watchOS 9, WidgetKit allows you to create accessory widgets that appear as complications in watchOS and as widgets on the Lock Screen on iPhone. Accessory widgets are a great opportunity to bring your iOS app content to Apple Watch and your watchOS app content to iPhone.

Note

You use WidgetKit to add support for Live Activities to your app. However, Live Activities aren't widgets, so you update their content differently. To learn more about Live Activities, see [ActivityKit](#).

To implement a widget, add a widget extension to your app. Configure the widget with a timeline provider, and use [SwiftUI](#) views to display the widget's content. The timeline provider tells WidgetKit when to update your widget's content.

```

graph LR
    Widget[Widget] --> Configuration[Configuration]
    Widget --> Provider[Provider]
    Widget --> ViewContent[View content]
    Provider --> TimelineEntry[Timeline entry]
    Provider --> Snapshot[Snapshot]
  
```

The screenshot shows the Apple Developer Documentation website for WidgetKit. The left sidebar contains a navigation tree for WidgetKit, including sections like Widget creation, Configuration, Timeline management, User interface, and Smart stacks. A 'Filter' button is at the bottom of the sidebar. The main content area displays detailed documentation for various WidgetKit components, such as the `Widget`, `WidgetBundle`, `StaticConfiguration`, `WidgetFamily`, `WidgetRenderingMode`, `IntentConfiguration`, `WidgetInfo`, and `Timeline`. A modal window is open over the content, showing the Swift code definitions for these protocols and structures.

```
protocol Widget
    The configuration and content of a widget to display on the Home screen or in Notification Center.

protocol WidgetBundle
    A container used to expose multiple widgets from a single widget extension.

struct StaticConfiguration
    An object describing the content of a widget that has no user-configurable options.

enum WidgetFamily
    Values that define the widget's size and shape.

struct WidgetRenderingMode
    Constants that indicate the rendering mode for a widget.
```

The screenshot shows a Mac OS X desktop environment with a browser window open to the Apple Developer Documentation at developer.apple.com. The page is for the `StaticConfiguration` API in WidgetKit.

Left Sidebar:

- WidgetKit
 - Widget creation
 - Creating a Widget Extension
 - Creating Lock Screen Widgets and ...
 - Migrating ClockKit complications to...
 - Building Widgets Using WidgetKit a...
 - Adding widgets to the Lock Screen ...
 - Fruta: Building a Feature-Rich App ...
 - Widget
 - WidgetBundle
 - StaticConfiguration
 - Creating a Widget Configuration
 - init<Provider>(kind: String, provi...)
 - var body: some WidgetConfigura...
 - func configurationDisplayName(l...
 - func configurationDisplayName(T...
 - func configurationDisplayName<...
 - Setting the Description
 - func description(LocalizedString...
 - func description(Text) -> some ...
 - func description<S>(S) -> some ...
 - Setting the Supported Families
 - func supportedFamilies([WidgetF...
 - Handling Background Network Req...
 - func onBackgroundURLSessionE...
 - func onBackgroundURLSessionE...
 - Type Aliases
 - StaticConfiguration.Body
 - Default Implementations
 - WidgetConfiguration Implementa...
 - WidgetFamily
 - WidgetRenderingMode
- Configurable Widgets
- Filter

Right Content Area:

Structure

StaticConfiguration

An object describing the content of a widget that has no user-configurable options.

(iOS 14.0+, iPadOS 14.0+, macOS 11.0+, Mac Catalyst 14.0+, watchOS 9.0+)

Declaration

```
struct StaticConfiguration<Content> where Content : View
```

Overview

The following example shows the configuration for a game widget that displays the status of the game.

```
struct GameStatusWidget: Widget {
    var body: some WidgetConfiguration {
        StaticConfiguration<
            kind: "com.mygame.game-status",
            provider: GameStatusProvider(),
        > { entry in
            GameStatusView(entry.gameStatus)
        }
        .configurationDisplayName("Game Status")
        .description("Shows an overview of your game status")
        .supportedFamilies([.systemSmall, .systemMedium, .systemLarge])
    }
}
```

Every widget has a unique kind, a string that you choose. You use this string to identify your widget when reloading its timeline with `WidgetCenter`.

The timeline provider is an object that determines the timeline for refreshing your widget. Providing future dates for updating your widget allows the system to optimize the refresh process.

The content closure contains the SwiftUI views that WidgetKit needs to render the widget. When WidgetKit invokes the content closure, it passes a timeline entry created by the widget provider's `getSnapshot(in:completion:)` or `getTimeline(in:completion:)` method.

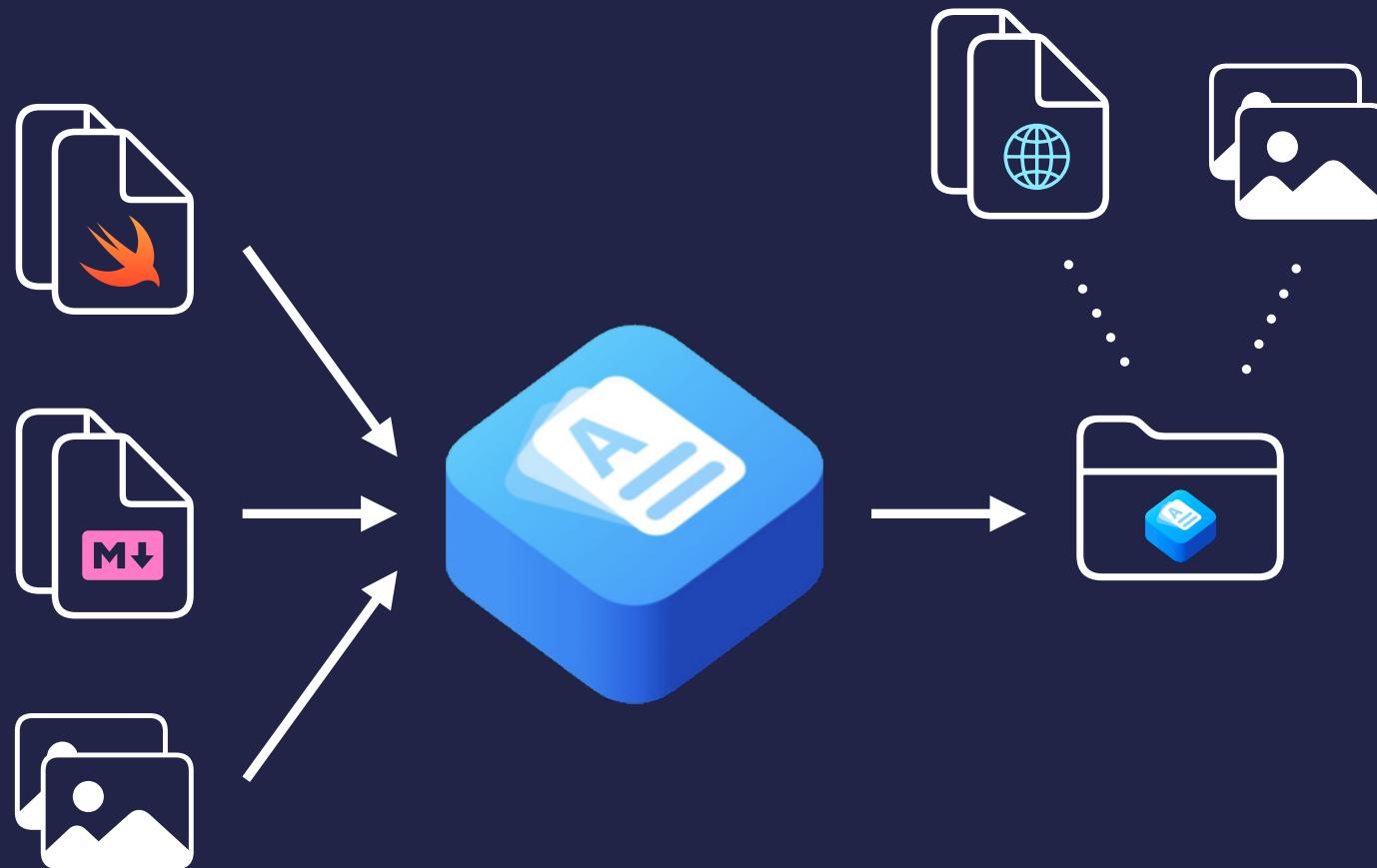
Modifiers let you specify the families your widget supports, and the details shown when users add or edit their widgets.

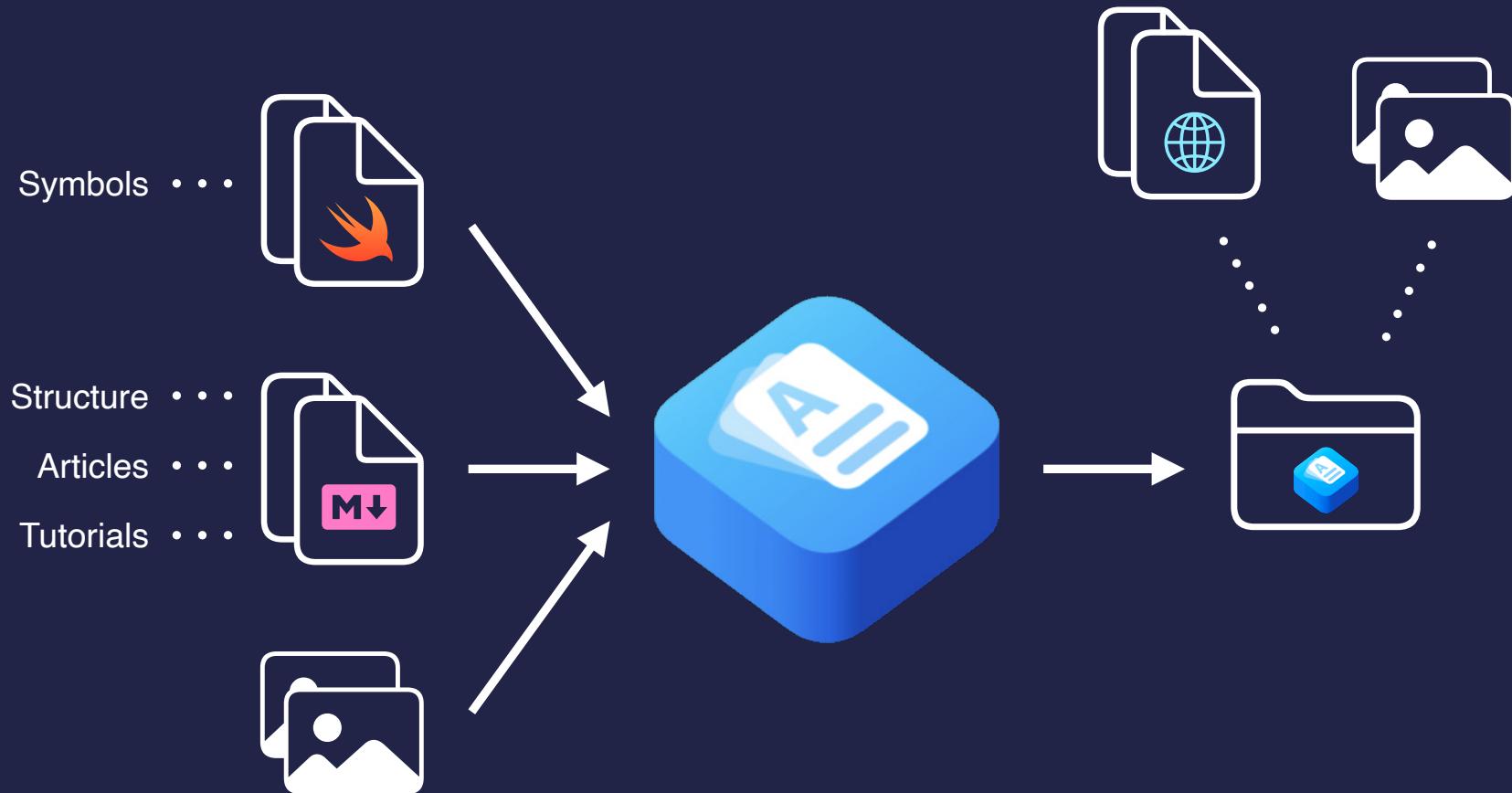
**Apple's documentation
is great.**

**Apple's documentation
is great when it exists.**



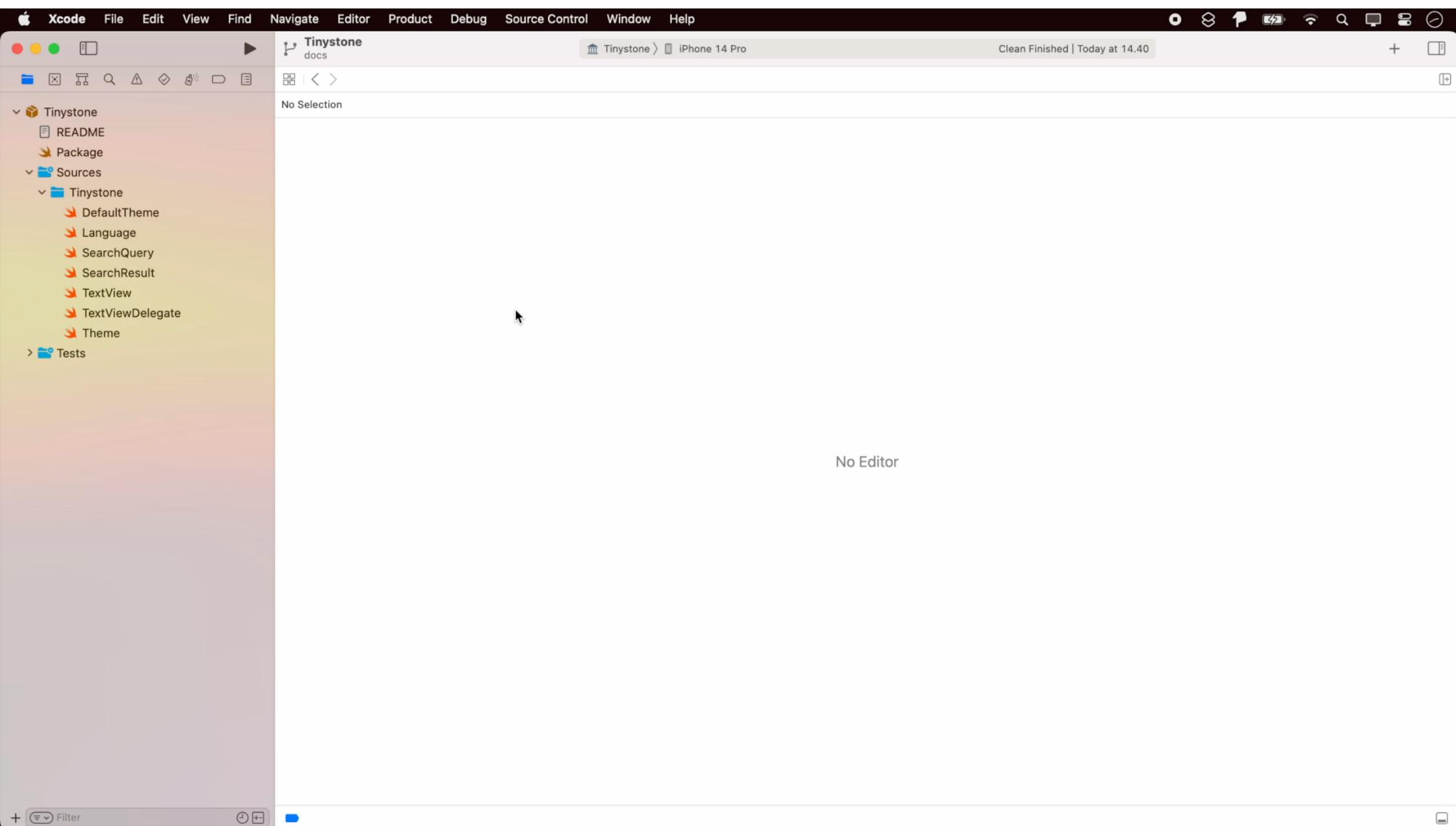


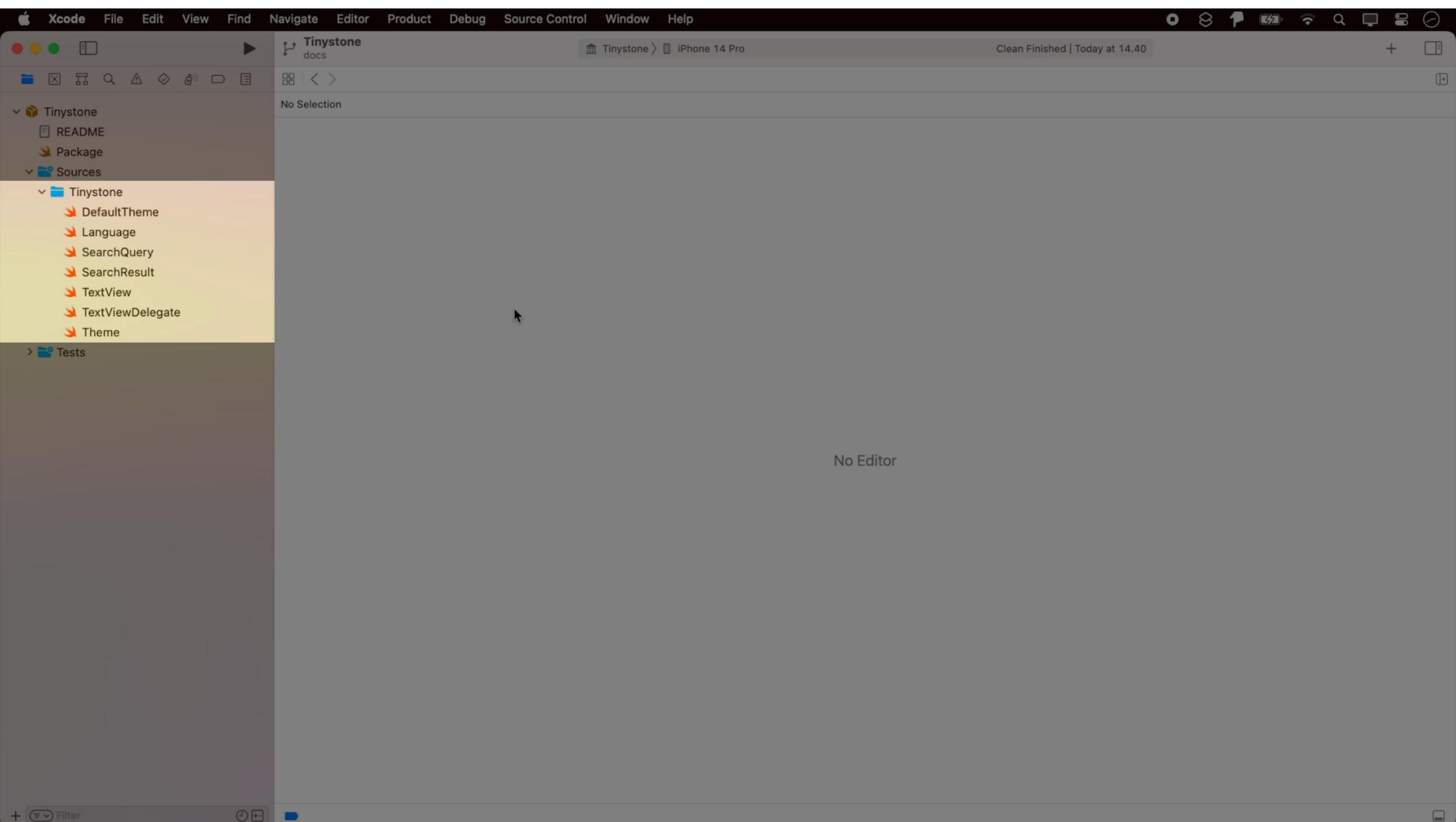


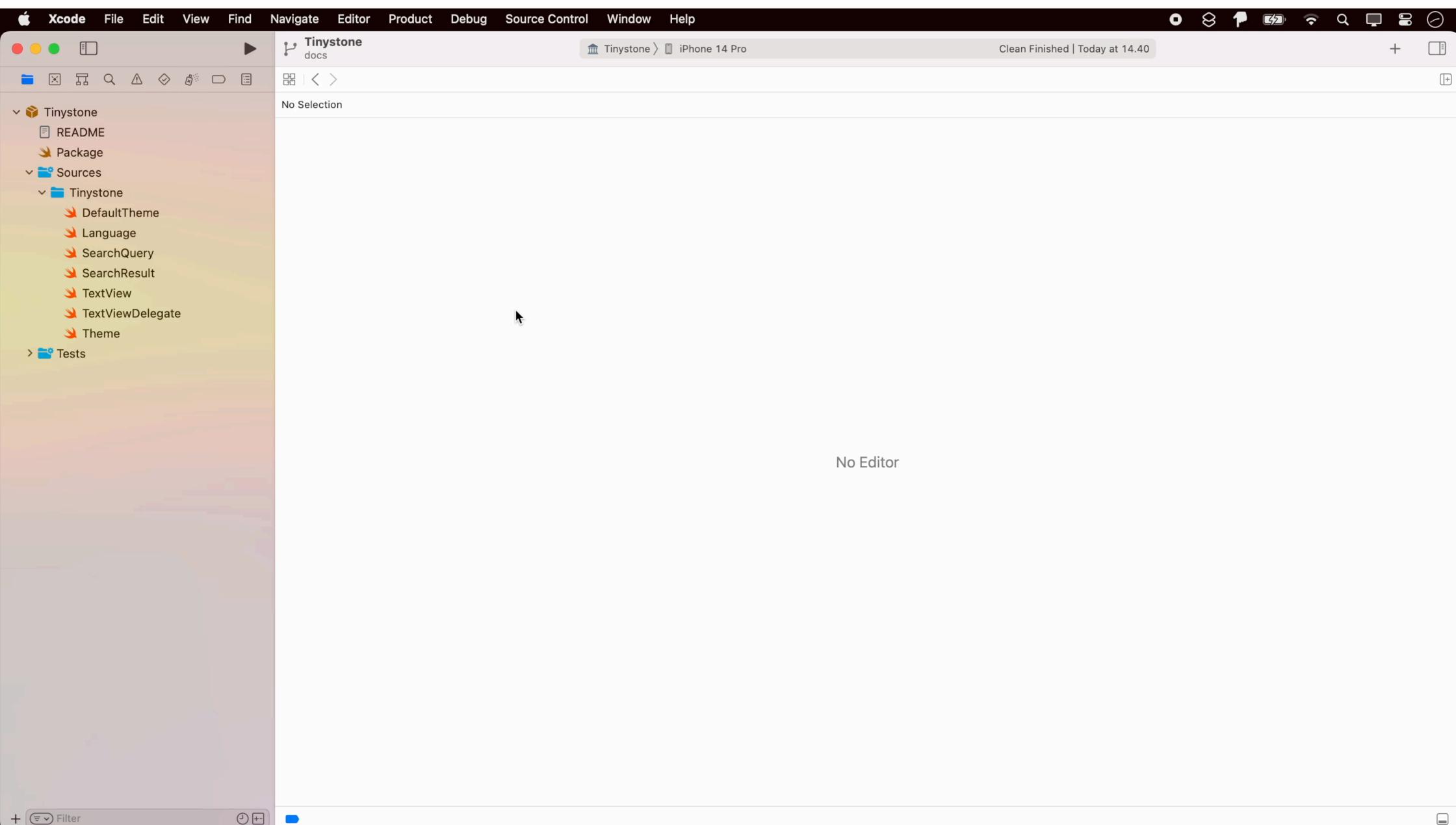


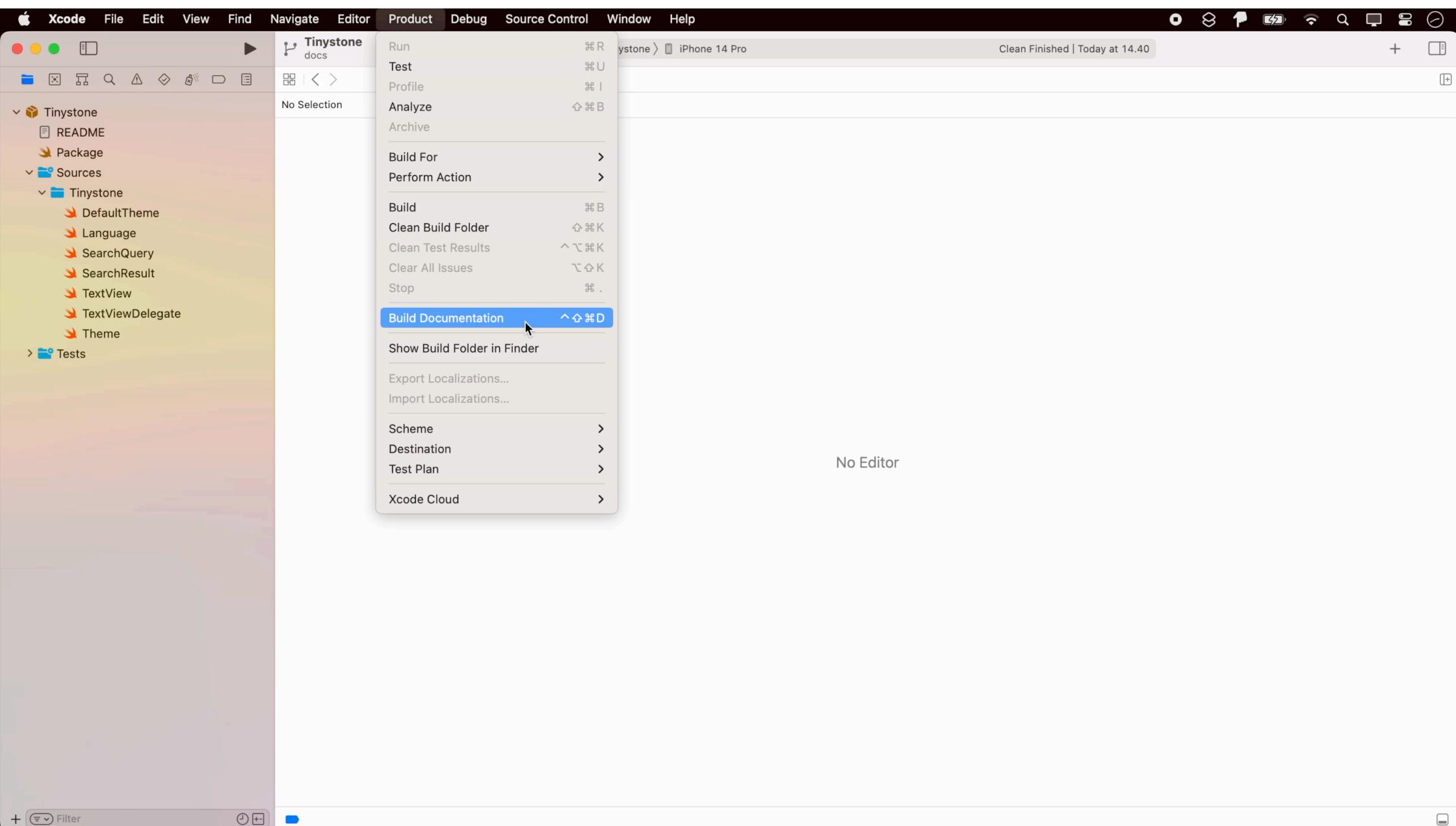
**Swift, Markdown, and resources go in,
documentation comes out.**

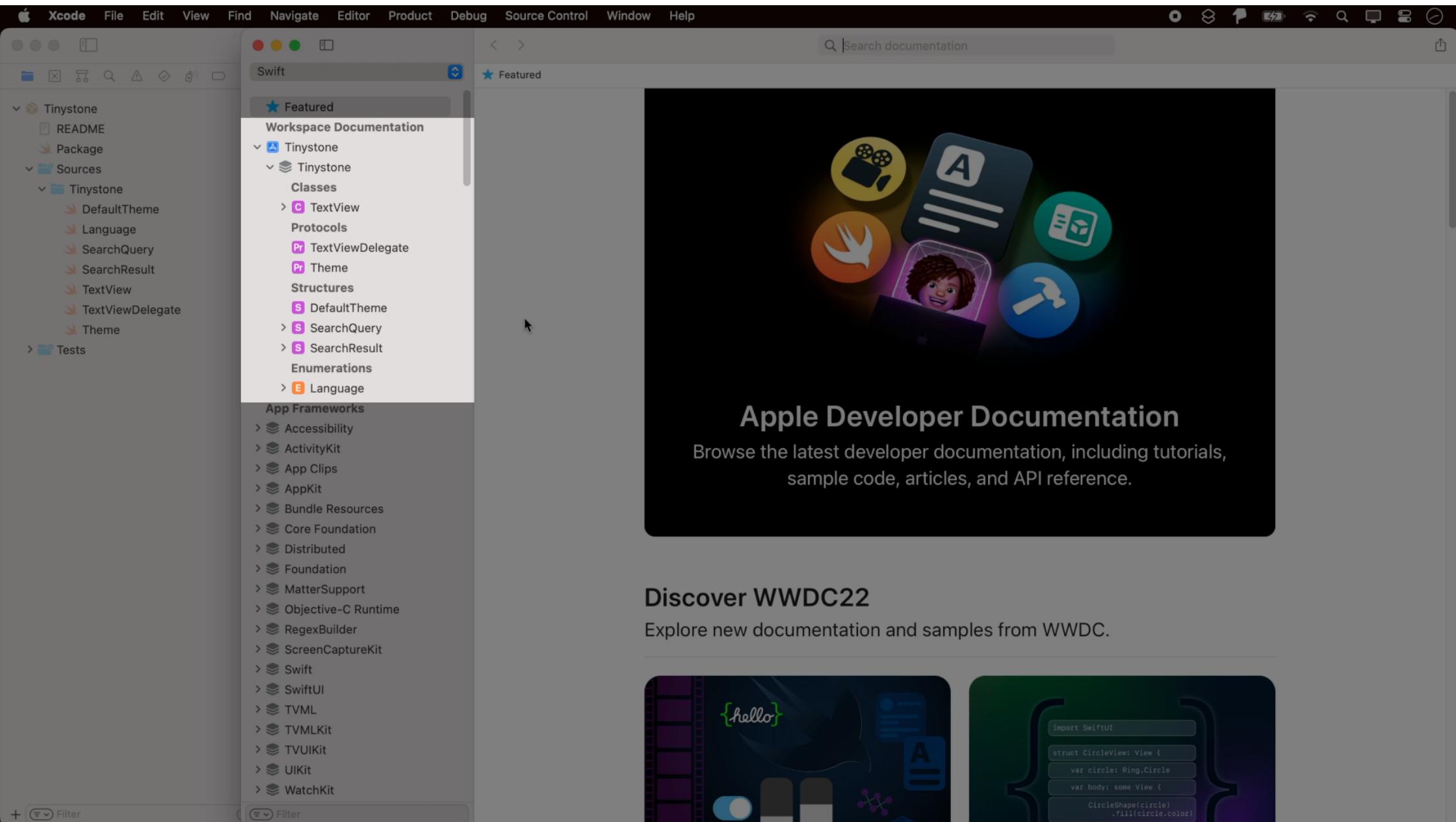












Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

Swift

Featured

Tinystone

Workspace Documentation

Tinystone

Classes

TextView

Protocols

TextDelegate

Theme

Structures

DefaultTheme

SearchQuery

SearchResult

TextView

TextViewDelegate

Theme

DefaultTheme

Language

SearchQuery

SearchResult

TextView

TextViewDelegate

Theme

Accessibility

ActivityKit

App Clips

AppKit

Bundle Resources

Core Foundation

Distributed

Foundation

MatterSupport

Objective-C Runtime

RegexBuilder

ScreenCaptureKit

Swift

SwiftUI

TVML

TVMLKit

TVUIKit

UIKit

WatchKit

Filter

Filter

Search documentation

Featured

Apple Developer Documentation

Browse the latest developer documentation, including tutorials, sample code, articles, and API reference.

Discover WWDC22

Explore new documentation and samples from WWDC.

```
import SwiftUI
```

```
struct CircleView: View {
```

```
    var circle: Ring.Circle
```

```
    var body: some View {
```

```
        CircleShape(circle)
```

```
.fill(circle.color)
```

The screenshot shows the Xcode interface with the Swift documentation open for the 'Tinystone' framework. The left sidebar displays the project structure with 'Tinystone' selected. The main pane shows the 'Tinystone' framework documentation, featuring a dark blue header with the word 'Tinystone' in white. Below the header, there's a section titled 'Topics' which is currently collapsed. Underneath, there are sections for 'Classes', 'Protocols', 'Structures', 'Enumerations', and 'App Frameworks'. Each section contains a list of corresponding Swift types (e.g., 'class TextView', 'protocol TextViewDelegate', etc.).

Swift

Featured

Workspace Documentation

Tinystone

Tinystone

Classes

> C TextView

Protocols

> Pr TextViewDelegate

> Pr Theme

Structures

> S DefaultTheme

> S SearchQuery

> S SearchResult

Enumerations

> E Language

App Frameworks

> Accessibility

> ActivityKit

> App Clips

> AppKit

> Bundle Resources

> Core Foundation

> Distributed

> Foundation

> MatterSupport

> Objective-C Runtime

> RegexBuilder

> ScreenCaptureKit

> Swift

> SwiftUI

> TVML

> TVMLKit

> TVUIKit

> UIKit

> WatchKit

Topics

Classes

class TextView

Protocols

protocol TextViewDelegate

protocol Theme

Structures

struct DefaultTheme

struct SearchQuery

struct SearchResult

Enumerations

enum Language



Describe Symbols



Describe Symbols



Add Structure



Describe Symbols



Add Structure



Add Supplemental
Content



Describe Symbols



Add Structure



Add Supplemental
Content



```
func search(for query: SearchQuery) -> [SearchResult]
```

Instance Method

search(for:)

Declaration

```
func search(for query: SearchQuery) -> [SearchResult]
```



```
func search(for query: SearchQuery) -> [SearchResult]
```

Instance Method

search(for:)

Declaration

```
func search(for query: SearchQuery) -> [SearchResult]
```



```
/// Search for the specified query.  
func search(for query: SearchQuery) -> [SearchResult]
```

Instance Method

search(for:)

Search for the specified query.

Declaration

```
func search(for query: SearchQuery) -> [SearchResult]
```



```
/// Search for the specified query.  
/// - Parameter query: Query to find matches for.  
/// - Returns: Results matching the query.  
func search(for query: SearchQuery) -> [SearchResult]
```

Instance Method

search(for:)

Search for the specified query.

Declaration

```
func search(for query: SearchQuery) -> [SearchResult]
```

Parameters

query

Query to find matches for.

Return Value

Results matching the query.



```
/// Search for the specified query.  
///  
/// ````swift  
/// let query = SearchQuery(text: "foo")  
/// let results = textView.search(for: query)  
/// ````  
///  
/// - Parameter query: Query to find matches for.  
/// - Returns: Results matching the query.  
  
func search(for query: SearchQuery) -> [SearchResult]
```

Instance Method

search(for:)

Search for the specified query.

Declaration

```
func search(for query: SearchQuery) -> [SearchResult]
```

Parameters

`query`
Query to find matches for.

Return Value

Results matching the query.

Discussion

```
let query = SearchQuery(text: "foo")  
let results = textView.search(for: query)
```



```
/// Search for the specified query.  
///  
/// The code below shows how a ``SearchQuery`` can  
/// be constructed and passed to ``search(for:)``.  
///  
/// swift  
/// let query = SearchQuery(text: "foo")  
/// let results = textView.search(for: query)  
///  
/// - Parameter query: Query to find matches for.  
/// - Returns: Results matching the query.  
  
func search(for query: SearchQuery) -> [SearchResult]
```

Instance Method

search(for:)

Search for the specified query.

Declaration

```
func search(for query: SearchQuery) -> [SearchResult]
```

Parameters

`query`
Query to find matches for.

Return Value

Results matching the query.

Discussion

The code below shows how a `SearchQuery` can be constructed and passed to `search(for:)`.

```
let query = SearchQuery(text: "foo")  
let results = textView.search(for: query)
```



```
///  
/// The code below shows how a ``SearchQuery`` can  
/// be constructed and passed to ``search(for:)``.  
///  
/// ````swift  
/// let query = SearchQuery(text: "foo")  
/// let results = textView.search(for: query)  
/// ````  
///  
/// > Note: A text of size N can be searched in time O(N).  
///  
/// - Parameter query: Query to find matches for.  
/// - Returns: Results matching the query.  
  
func search(for query: SearchQuery) -> [SearchResult]
```

Search for the specified query.

Declaration

```
func search(for query: SearchQuery) -> [SearchResult]
```

Parameters

query
Query to find matches for.

Return Value

Results matching the query.

Discussion

The code below shows how a `SearchQuery` can be constructed and passed to `search(for:)`.

```
let query = SearchQuery(text: "foo")  
let results = textView.search(for: query)
```

Note

A text of size N can be searched in time O(N).

TextView.swift

```
/// The code below shows how a ``SearchQuery`` can
/// be constructed and passed to ``search(for:)``.
///
/// ``swift
/// let query = SearchQuery(text: "foo")
/// let results = textView.search(for: query)
/// ```
///
/// > Note: A text of size N can be searched in time  $O(N)$ .
///
/// - Parameter query: Query to find matches for.
/// - Returns: Results matching the query.
@available(iOS 16, macOS 13, *)
func search(for query: SearchQuery) -> [SearchResult]
```

Instance Method

search(for:)

Search for the specified query.

iOS 16.0+ macOS 13.0+

Declaration

```
func search(for query: SearchQuery) -> [SearchResult]
```

Parameters

query
Query to find matches for.

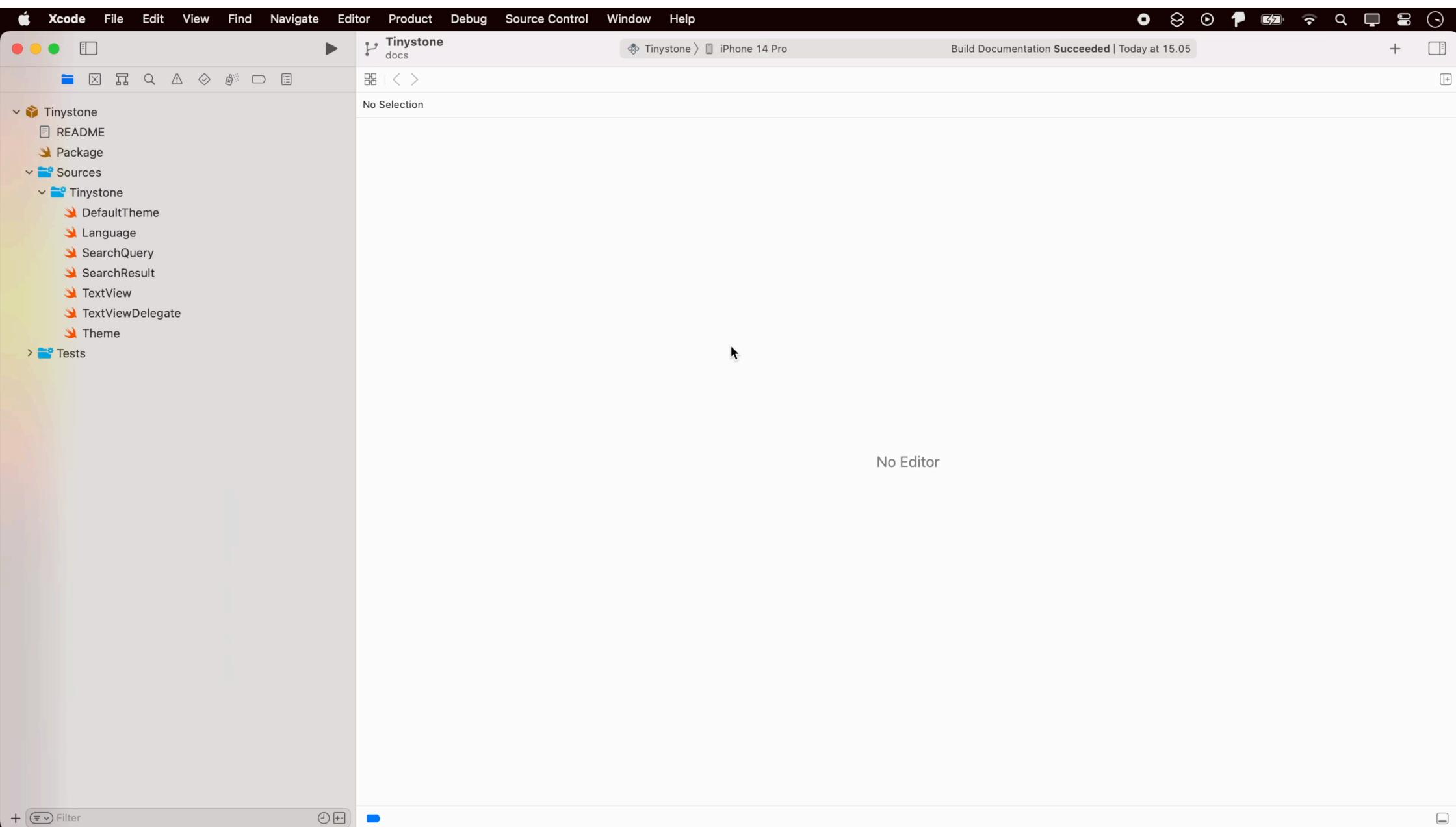
Return Value

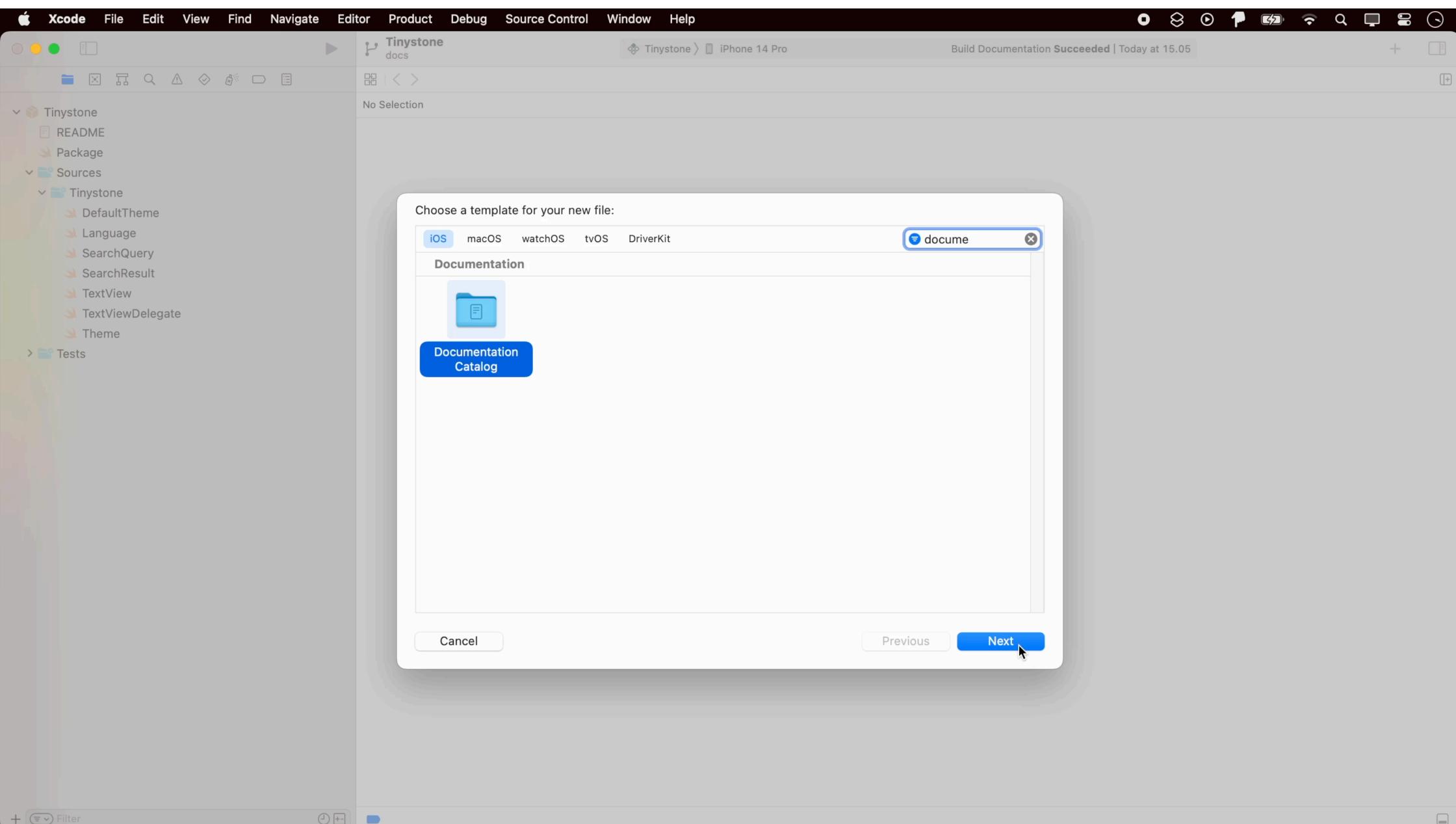
Results matching the query.

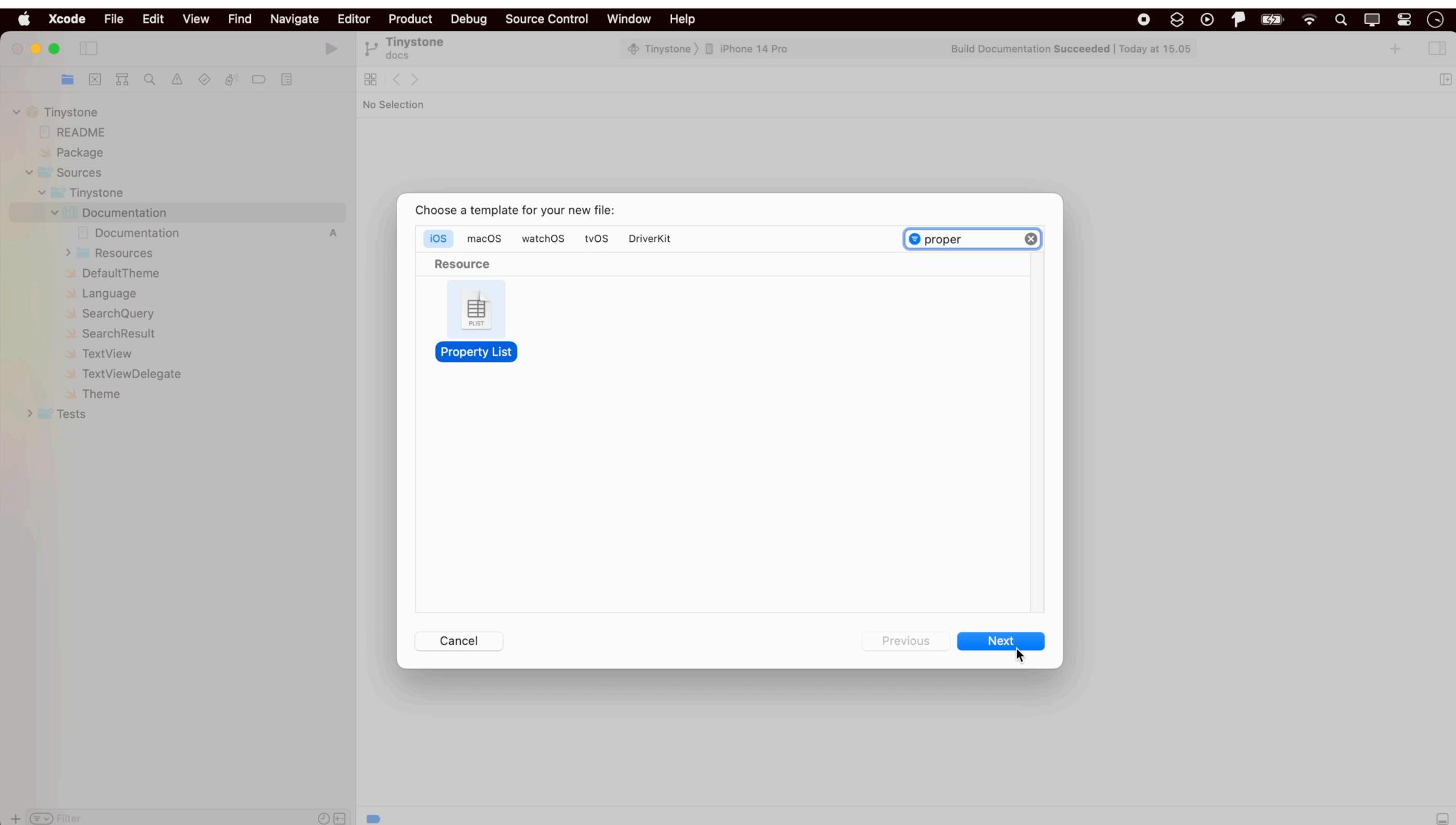
Discussion

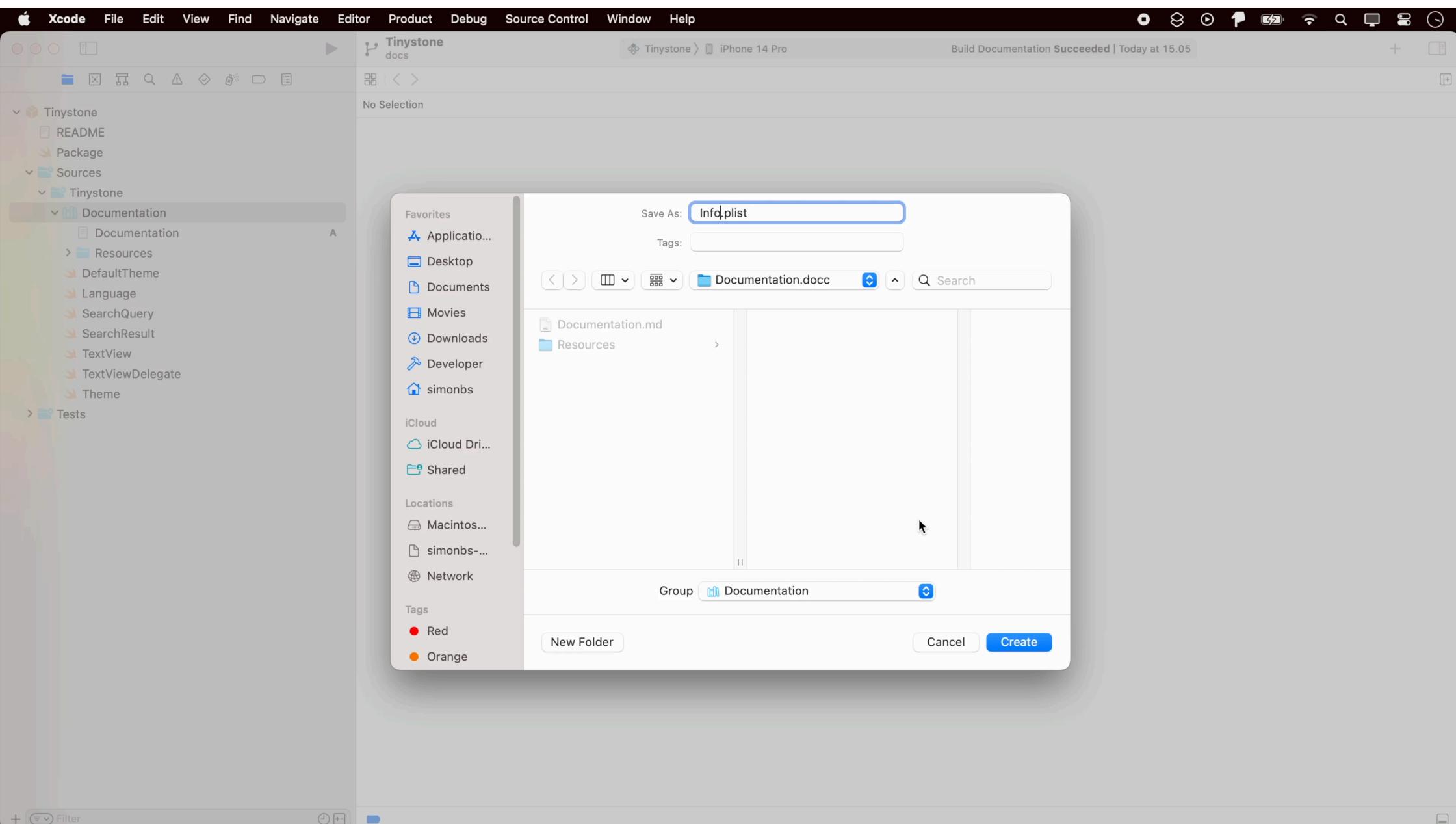
The code below shows how a `SearchQuery` can be constructed and passed to `search(for:)`.

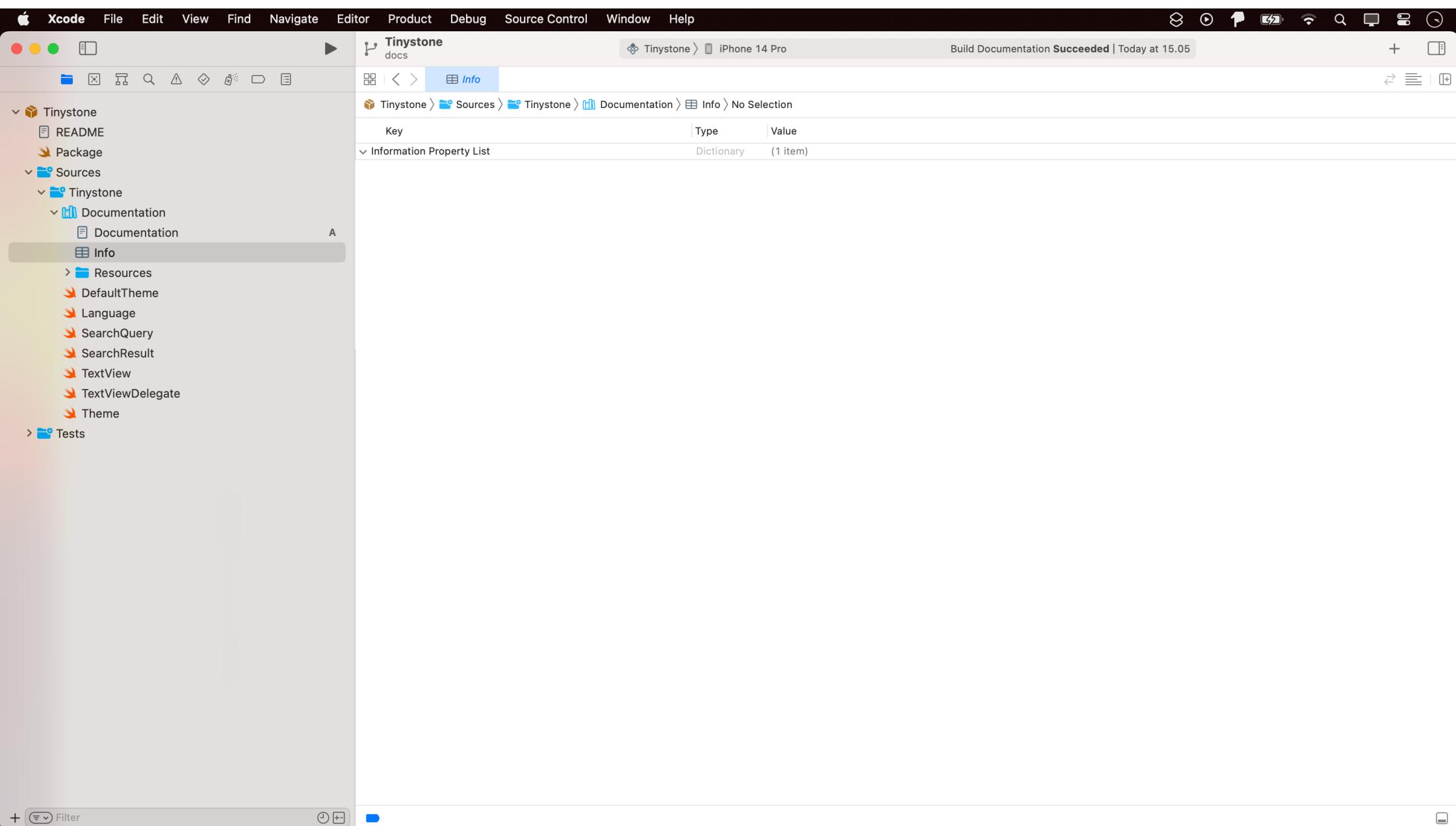
```
let query = SearchQuery(text: "foo", matchMethod: .contains)
let results = textView.search(for: query)
```











▶ **Tinystone**
docs

◆ Tinystone ▶ iPhone 14 Pro

| < > Info

Tinystone > Sources > Tinystone > Documentation > Info > No Selection

Key	Type	Value
Information Property List	Dictionary	(1 item)
CDAppleDefaultAvailability	Dictionary	(1 item)
Tinystone	Array	(2 items)
Item 0	Dictionary	(2 items)
name	String	iOS
version	String	14.0
Item 1	Dictionary	(2 items)
name	String	macOS
version	String	12.0

Tinystone
docs

Tinystone > iPhone 14 Pro

Info

Tinystone > Sources > Tinystone > Documentation > Info > No Selection

Key	Type	Value
Information Property List	Dictionary	(1 item)
CDAppleDefaultAvailability	Dictionary	(1 item)
Tinystone	Array	(2 items)
Item 0	Dictionary	(2 items)
name	String	iOS
version	String	14.0
Item 1	Dictionary	(2 items)
name	String	macOS
version	String	12.0

The screenshot shows the Xcode Documentation viewer open to the `TextView` class documentation. The sidebar on the left lists various Swift workspace documentation, including `Tinystone`, `Accessibility`, `ActivityKit`, `App Clips`, `AppKit`, `Bundle Resources`, `Core Foundation`, `Distributed`, `Foundation`, `MatterSupport`, `Objective-C Runtime`, `RegexBuilder`, `ScreenCaptureKit`, `Swift`, `SwiftUI`, `TVML`, `TVMLKit`, `TVUIKit`, `UIKit`, `WatchKit`, and `watchOS apps`. The main content area displays the `TextView` class documentation, which is described as a type similar to `UITextView` with features commonly found in code editors. It includes support for iOS 14.0+, macOS 12.0+, and Mac Catalyst 14.0+. The declaration is shown as `final class TextView`. The overview section states that `TextView` is a performant implementation of a text view with features such as showing line numbers, searching for text and replacing results, syntax highlighting, showing invisible characters and more. It also notes that to enable syntax highlighting, both a [Language](#) and a [Theme](#) must be provided. The topics section is currently empty.

Swift

Featured

Workspace Documentation

✓ Tinystone

Tinystone

App Frameworks

Accessibility

ActivityKit

App Clips

AppKit

Bundle Resources

Core Foundation

Distributed

Foundation

MatterSupport

Objective-C Runtime

RegexBuilder

ScreenCaptureKit

Swift

SwiftUI

TVML

TVMLKit

TVUIKit

UIKit

WatchKit

watchOS apps

App Services

Accounts

Address Book

Address Book UI

AdServices

AdSupport

App Intents

Application Services

Filter

Search documentation

Tinystone > Tinystone > TextView

Class

TextView

A type similiar to `UITextView` with features commonly found in code editors.

iOS 14.0+ macOS 12.0+ Mac Catalyst 14.0+

Declaration

```
final class TextView
```

Overview

`TextView` is a performant implementation of a text view with features such as showing line numbers, searching for text and replacing results, syntax highlighting, showing invisible characters and more.

To enable syntax highlighting in the text view, you must provide both a [Language](#) and a [Theme](#).

Topics

TextView.swift

```
/// The code below shows how a ``SearchQuery`` can
/// be constructed and passed to ``search(for:)``.
///
/// ``swift
/// let query = SearchQuery(text: "foo")
/// let results = textView.search(for: query)
/// ```
///
/// > Note: A text of size N can be searched in time  $O(N)$ .
///
/// - Parameter query: Query to find matches for.
/// - Returns: Results matching the query.
@available(iOS 16, macOS 13, *)
func search(for query: SearchQuery) -> [SearchResult]
```

Instance Method

search(for:)

Search for the specified query.

iOS 16.0+ macOS 13.0+

Declaration

```
func search(for query: SearchQuery) -> [SearchResult]
```

Parameters

query
Query to find matches for.

Return Value

Results matching the query.

Discussion

The code below shows how a `SearchQuery` can be constructed and passed to `search(for:)`.

```
let query = SearchQuery(text: "foo", matchMethod: .contains)
let results = textView.search(for: query)
```

```
TextView.swift
// DE CONSTRUCTED AND PASSED TO SEARCH(TOTL) ...

/// 
/// ````swift
/// let query = SearchQuery(text: "foo")
/// let results = textView.search(for: query)
/// ````

///
/// > Note: A text of size N can be searched in time O(N).

///
/// I wasted a @#$%&! load of hours on this @#$%&! 😱

///
/// - Parameter query: Query to find matches for.
/// - Returns: Results matching the query.

@available(iOS 16, macOS 13, *)
func search(for query: SearchQuery) -> [SearchResult]

```

Declaration

```
func search(for query: SearchQuery) -> [SearchResult]
```

Parameters

`query`
Query to find matches for.

Return Value

Results matching the query.

Discussion

The code below shows how a `SearchQuery` can be constructed and passed to `search(for:)`.

```
let query = SearchQuery(text: "foo", matchMethod: .contains)
let results = textView.search(for: query)
```

Note

A string of size N can be searched in time O(N).

I wasted a @#\$%&! load of hours on this @#\$%&! 😱

 **TextView.swift**

```
/// Be constructed and passed to search(for:) . . .

/// ````swift

/// let query = SearchQuery(text: "foo")
/// let results = textView.search(for: query)

/// ````

/// > Note: A text of size N can be searched in time O(N).

/// - Parameter query: Query to find matches for.
/// - Returns: Results matching the query.

@available(iOS 16, macOS 13, *)
func search(for query: SearchQuery) -> [SearchResult] {
    // I wasted a @#$%&! load of hours on this @#$%&!
}
```

Instance Method

search(for:)

Search for the specified query.

iOS 16.0+ macOS 13.0+

Declaration

```
func search(for query: SearchQuery) -> [SearchResult]
```

Parameters

query
Query to find matches for.

Return Value

Results matching the query.

Discussion

The code below shows how a `SearchQuery` can be constructed and passed to `search(for:)`.

```
let query = SearchQuery(text: "foo", matchMethod: .contains)
let results = textView.search(for: query)
```



Describe Symbols



Add Structure



Add Supplemental
Content



Describe Symbols



Add Structure



Add Supplemental
Content

Framework

Tinystone

Performant code editor with syntax highlighting, line numbers, and invisible characters.



Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.



Topics

Essentials

- ↳ [Meet Tinystone](#)
Explore Tinystone, the performant code editor with syntax highlighting, line numbers, and invisible characters through a series of interactive tutorials.

Class

```
class TextView
```

Protocol

```
protocol TextViewDelegate
```

The methods for receiving editing-related messages for the text view.

Syntax Highlighting

```
enum Language
```

Language that can be used to syntax highlight text in the text view.

```
protocol Theme
```

Fonts and colors to be used by a TextView.

```
struct DefaultTheme
```

The theme used by a text view when no other theme has been specified.

Find and Replace

```
struct SearchQuery
```

Query to search for in the text view.

```
struct SearchResult
```

A match returned by performing a search query.

Landing Page

Class

TextView

A type similar to UITextView with features commonly found in code editors.

Declaration

```
final class TextView
```

Overview

TextView is a performant implementation of a text view with features such as showing line numbers, searching for text and replacing results, syntax highlighting, showing invisible characters and more.

To enable syntax highlighting in the text view, you must provide both a [Language](#) and a [Theme](#).

Topics

Essentials

```
init(frame: CGRect)
```

Create a new text view.

```
var text: String
```

The text that the text view displays.

```
var textViewDelegate: TextViewDelegate?
```

The object to receive editing callbacks.

```
protocol TextViewDelegate
```

The methods for receiving editing-related messages for the text view.

Text Layout

```
var isLineWrappingEnabled: Bool
```

When line wrapping is disabled, users can scroll the text view horizontally to see the entire line.

```
var showInvisibleCharacters: Bool
```

Enable to show invisible characters.

```
var lineHeightMultiplier: CGFloat
```

The line-height is multiplied with the value.

```
var kern: CGFloat
```

The number of points by which to adjust kern.

Syntax Highlighting

```
var language: Language
```

The language used for syntax highlighting text in the text view.

```
var theme: Theme
```

Fonts and colors used when syntax highlighting text.

Gutter

```
var showLineNumbers: Bool
```

Enable to show line numbers in the gutter.

```
var gutterWidth: CGFloat
```

Width of the gutter.

Extensions

Framework

Tinystone

Performant code editor with syntax highlighting, line numbers, and invisible characters.



Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.



Topics

Essentials

[Meet Tinystone](#)

Explore Tinystone, the performant code editor with syntax highlighting, line numbers, and invisible characters through a series of interactive tutorials.

`class TextView`

A type similar to UITextView with features commonly found in code editors.



Topics

Essentials

[Meet Tinystone](#)

Explore Tinystone, the performant code editor with syntax highlighting, line numbers, and invisible characters through a series of interactive tutorials.

`class TextView`

A type similar to UITextView with features commonly found in code editors.

`protocol TextViewDelegate`

The methods for receiving editing-related messages for the text view.

Syntax Highlighting

`enum Language`

Language that can be used to syntax highlight text in the text view.

`protocol Theme`

Fonts and colors to be used by a TextView.

`struct DefaultTheme`

The theme used by a text view when no other theme has been specified.

Find and Replace

`struct SearchQuery`

Query to search for in the text view.

`struct SearchResult`

A match returned by performing a search query.

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

Swift

Tinystone > Tinystone

Search documentation

Featured

Workspace Documentation

Tinystone

App Frameworks

Accessibility

ActivityKit

App Clips

AppKit

Bundle Resources

Core Foundation

Distributed

Foundation

MatterSupport

Objective-C Runtime

RegexBuilder

ScreenCaptureKit

Swift

SwiftUI

TVML

TVMLKit

TVUIKit

UIKit

WatchKit

watchOS apps

App Services

Accounts

Address Book

Address Book UI

AdServices

AdSupport

App Intents

Application Services

Automatic Assessment Co...

Background Assets

Topics

Classes

class `TextView`

A type similar to UITextView with features commonly found in code editors.

Protocols

protocol `TextViewDelegate`

The methods for receiving editing-related messages for the text view.

protocol `Theme`

Fonts and colors to be used by a TextView.

Structures

struct `DefaultTheme`

The theme used by a text view when no other theme has been specified.

+

Filter

Filter

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

Swift

Tinystone > Tinystone

Search documentation

Featured

Workspace Documentation

Tinystone

App Frameworks

Accessibility

ActivityKit

App Clips

AppKit

Bundle Resources

Core Foundation

Distributed

Foundation

MatterSupport

Objective-C Runtime

RegexBuilder

ScreenCaptureKit

Swift

SwiftUI

TVML

TVMLKit

TVUIKit

UIKit

WatchKit

watchOS apps

App Services

Accounts

Address Book

Address Book UI

AdServices

AdSupport

App Intents

Application Services

Automatic Assessment Co...

Background Assets

Framework

Tinystone

iOS 14.0+ macOS 12.0+ Mac Catalyst 14.0+

Topics

Classes

class `TextView`

A type similar to UITextView with features commonly found in code editors.

Protocols

protocol `TextViewDelegate`

The methods for receiving editing-related messages for the text view.

protocol `Theme`

Fonts and colors to be used by a TextView.

Structures

struct `DefaultTheme`

The theme used by a text view when no other theme has been specified.

A screenshot of Xcode showing the Swift documentation interface for a project named "Tinystone".

The left sidebar shows the project structure:

- Tinystone (group)
- README
- Package
- Sources (group)
- Tinystone (group)
 - Documentation
 - Documentation
 - Info
 - Resources
 - DefaultTheme
 - Language
 - SearchQuery
 - SearchResult
 - TextView
 - TextViewDelegate
 - Theme
- Tests

The main pane displays the Swift documentation for the "Tinystone" framework. The title "Tinystone" is prominently displayed, along with compatibility notes for iOS 14.0+, macOS 12.0+, and Mac Catalyst 14.0+. The documentation is organized into sections:

- Topics**
- Classes**
 - `class TextView`

A type similar to UITextView with features commonly found in code editors.
- Protocols**
 - `protocol TextViewDelegate`

The methods for receiving editing-related messages for the text view.
 - `protocol Theme`

Fonts and colors to be used by a TextView.
- Structures**
 - `struct DefaultTheme`

The theme used by a text view when no other theme has been specified.

````

Summary

Overview

Text

Topics

Groups

- ``Symbol``



````

Summary

Overview

Text

Topics

Groups

- ``Symbol``

``Tinystone``

Summary

Overview

Text

Topics

Groups

- ``Symbol``

Framework

Tinystone

Summary

Overview

Text

Topics

Classes

class `TextView`

A type similiar to UITextView with features commonly found in code editors.

Protocols

protocol `Theme`

Fonts and colors to be used by a TextView.

Structures

struct `DefaultTheme`

The theme used by a text view when no other theme has been specified.

struct `SearchQuery`

Query to search for in the text view.

``Tinystone``

Performant code editor with syntax highlighting, line numbers, and invisible characters.

Overview

Text

Topics

Groups

- ``Symbol``

Framework

Tinystone

Performant code editor with syntax highlighting, line numbers, and invisible characters.

Overview

Text

Topics

Classes

class `TextView`

A type similar to UITextView with features commonly found in code editors.

Protocols

protocol `Theme`

Fonts and colors to be used by a TextView.

Structures

struct `DefaultTheme`

The theme used by a text view when no other theme has been specified.

struct `SearchQuery`

Query to search for in the text view.

``Tinystone``

Performant code editor with syntax highlighting, line numbers, and invisible characters.

Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.

Topics

Groups

- ``Symbol``

Framework

Tinystone

Performant code editor with syntax highlighting, line numbers, and invisible characters.

Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.

Topics

Classes

class `TextView`

A type similiar to `UITextView` with features commonly found in code editors.

Protocols

protocol `Theme`

Fonts and colors to be used by a `TextView`.

``Tinystone``

Performant code editor with syntax highlighting, line numbers, and invisible characters.

Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.

Topics

Groups

- ``Symbol``

The screenshot shows the Xcode interface with the following details:

- Menu Bar:** Xcode, File, Edit, View, Find, Navigate, Editor, Product, Debug, Source Control, Window, Help.
- Toolbar:** Standard Xcode toolbar with icons for file operations.
- Project Navigator:** Shows the project structure:
 - Tinystone (root folder)
 - README
 - Package
 - Sources
 - Tinystone
 - Documentation (selected)
 - Resources
 - DefaultTheme
 - Language
 - SearchQuery
 - SearchResult
 - TextView
 - TextViewDelegate
 - Theme
 - Tests
- Document Area:** Displays the contents of the selected Documentation file.

Build Documentation Succeeded | Today at 13:04

Documentation

Tinystone > Sources > Tinystone > Documentation > Documentation > No Selection

```
1 # ``Tinystone``  
2  
3 Performant code editor with syntax highlighting, line numbers, and invisible  
   characters.  
4  
5 ## Overview  
6  
7 Tinystone provides a text view with support for features commonly found in  
   code editors. This includes the following list of features.  
8  
9 - Syntax highlight the text.  
10 - Show line numbers in the gutter.  
11 - Showing invisible characters such as tabs, spaces, and line breaks.  
12 - Adjust the height of lines.  
13  
14 ## Topics  
15  
16 ### Group  
17  
18 - ``Symbol``  
19
```
- Bottom Status Bar:** Filter, Line: 12 Col: 30.

``Tinystone``

Performant code editor with syntax highlighting, line numbers, and invisible characters.

Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.

Topics

Groups

- ``Symbol``

``Tinystone``

Performant code editor with syntax highlighting, line numbers, and invisible characters.

Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.

Topics

Groups

- ``Symbol``

Framework

Tinystone

Performant code editor with syntax highlighting, line numbers, and invisible characters.

Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.

Topics

Classes

class `TextView`

A type similiar to UITextView with features commonly found in code editors.

Protocols

protocol `Theme`

Fonts and colors to be used by a TextView.

``Tinystone``

Performant code editor with syntax highlighting, line numbers, and invisible characters.

Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.

![Hero image showing a happy stone in fluorescent colors]
(hero.png)

Topics

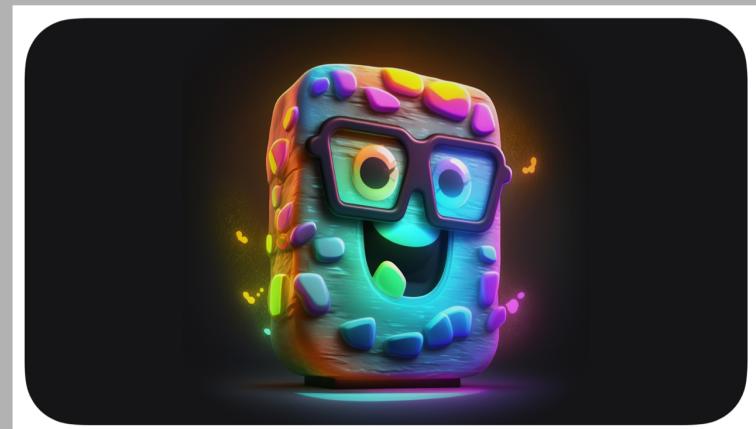
Groups

invisible characters.

Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.



Topics

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

Swift

Tinystone > Tinystone

Search documentation

Featured

Workspace Documentation

Tinystone

Tinystone

Classes

TextView

Protocols

TextViewDelegate

Theme

Structures

DefaultTheme

SearchQuery

SearchResult

TextView

TextViewDelegate

Theme

Tests

Resources

hero

DefaultTheme

Language

SearchQuery

SearchResult

TextView

TextViewDelegate

Theme

Accessibility

ActivityKit

App Clips

AppKit

Bundle Resources

Core Foundation

Distributed

Foundation

MatterSupport

Objective-C Runtime

RegexBuilder

ScreenCaptureKit

Swift

SwiftUI

TVML

TVMLKit

TVUIKit

UIKit

WatchKit

Framework

Tinystone

Performant code editor with syntax highlighting, line numbers, and invisible characters.

Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.



Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

Swift

Tinystone > Tinystone

Search documentation

Featured

Workspace Documentation

Tinystone

Tinystone

Classes

TextView

Protocols

TextViewDelegate

Theme

Structures

DefaultTheme

SearchQuery

SearchResult

TextView

TextViewDelegate

Language

App Frameworks

Accessibility

ActivityKit

App Clips

AppKit

Bundle Resources

Core Foundation

Distributed

Foundation

MatterSupport

Objective-C Runtime

RegexBuilder

ScreenCaptureKit

Swift

SwiftUI

TVML

TVMLKit

TVUIKit

UIKit

WatchKit

Documentation

hero

DefaultTheme

Language

SearchQuery

SearchResult

TextView

TextViewDelegate

Theme

Tests

Documentation

README

Package

Sources

Tinystone

Documentation

Resources

hero

DefaultTheme

Language

SearchQuery

SearchResult

TextView

TextViewDelegate

Theme

Tests

Framework

Tinystone

Performant code editor with syntax highlighting, line numbers, and invisible characters.

Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.



The screenshot shows the Xcode Documentation browser interface. The left sidebar displays the project structure for 'Tinystone'. The main pane shows the Swift documentation for the 'Tinystone' package, specifically the 'Documentation' folder.

Classes

`class TextView`
A type similar to UITextView with features commonly found in code editors.

Protocols

`protocol TextViewDelegate`
The methods for receiving editing-related messages for the text view.

`protocol Theme`
Fonts and colors to be used by a TextView.

Structures

`struct DefaultTheme`
The theme used by a text view when no other theme has been specified.

`struct SearchQuery`
Query to search for in the text view.

`struct SearchResult`
A match returned by performing a search query.

Enumerations

`enum Language`
Language that can be used to syntax highlight text in the text view.

The screenshot shows a documentation editor window with a dark theme. At the top, there's a toolbar with icons for Control, Window, Help, and a search bar labeled "Search documentation". The main area contains several sections:

- Classes**
class `TextView`
A type similar to UITextView with features commonly found in code editors.
- Protocols**
protocol `TextViewDelegate`
The methods for receiving editing-related messages for the text view.
protocol `Theme`
Fonts and colors to be used by a TextView.
- Structures**
struct `DefaultTheme`
The theme used by a text view when no other theme has been specified.
struct `SearchQuery`
Query to search for in the text view.
struct `SearchResult`
A match returned by performing a search query.
- Enumerations**
enum `Language`
Language that can be used to syntax highlight text in the text view.

Documentation.md

``Tinystone``

Performant code editor with syntax hi

Overview

Tinystone provides a text view with su
following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as
- Adjust the height of lines.

![Hero image showing a happy stone in

Topics

Groups

- ``Symbol``

Control Window Help

Tinystone

Search documentation

Classes

`class TextView`

A type similar to UITextView with features commonly found in code editors.

Protocols

`protocol TextViewDelegate`

The methods for receiving editing-related messages for the text view.

`protocol Theme`

Fonts and colors to be used by a TextView.

Structures

`struct DefaultTheme`

The theme used by a text view when no other theme has been specified.

`struct SearchQuery`

Query to search for in the text view.

`struct SearchResult`

A match returned by performing a search query.

Enumerations

`enum Language`

Language that can be used to syntax highlight text in the text view.

Documentation.md

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as
- Adjust the height of lines.

![Hero image showing a happy stone in

Topics

Groups

- ``Symbol``

Classes

class TextView

A type similar to UITextView with features commonly found in code editors.

Protocols

protocol TextViewDelegate

The methods for receiving editing-related messages for the text view.

protocol Theme

Fonts and colors to be used by a TextView.

Structures

struct DefaultTheme

The theme used by a text view when no other theme has been specified.

struct SearchQuery

Query to search for in the text view.

struct SearchResult

A match returned by performing a search query.

Enumerations

enum Language

Language that can be used to syntax highlight text in the text view.



Documentation.md

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as
- Adjust the height of lines.

![Hero image showing a happy stone in

Topics

Find and Replace

Search documentation

Classes

class TextView

A type similar to UITextView with features commonly found in code editors.

Protocols

protocol TextViewDelegate

The methods for receiving editing-related messages for the text view.

protocol Theme

Fonts and colors to be used by a TextView.

Structures

struct DefaultTheme

The theme used by a text view when no other theme has been specified.

struct SearchQuery

Query to search for in the text view.

struct SearchResult

A match returned by performing a search query.

Enumerations

enum Language

Language that can be used to syntax highlight text in the text view.



Documentation.md

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as
- Adjust the height of lines.

![Hero image showing a happy stone in

Topics

Syntax Highlighting

Find and Replace

Classes

class TextView

A type similar to UITextView with features commonly found in code editors.

Protocols

protocol TextViewDelegate

The methods for receiving editing-related messages for the text view.

protocol Theme

Fonts and colors to be used by a TextView.

Structures

struct DefaultTheme

The theme used by a text view when no other theme has been specified.

struct SearchQuery

Query to search for in the text view.

struct SearchResult

A match returned by performing a search query.

Enumerations

enum Language

Language that can be used to syntax highlight text in the text view.



Documentation.md

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as
- Adjust the height of lines.

![Hero image showing a happy stone in

Topics

Essentials

Syntax Highlighting

Find and Replace

The screenshot shows a documentation application interface with a search bar at the top. Below the search bar, there are several sections: **Classes**, **Protocols**, **Structures**, **Enumerations**, and **Documentation**. Each section contains one or more API cards. For example, the **Classes** section has a card for `TextView` which describes it as a type similar to `UITextView` with features found in code editors. The **Protocols** section has cards for `TextViewDelegate` and `Theme`. The **Structures** section has cards for `DefaultTheme`, `SearchQuery`, and `SearchResult`. The **Enumerations** section has a card for `Language`. The **Documentation** section has a card for `Documentation.md` which lists features like syntax highlighting, line numbers, and invisible characters.

Control Window Help

Search documentation

Documentation

Classes

`class TextView`
A type similiar to UITextView with features commonly found in code editors.

Protocols

`protocol TextViewDelegate`
The methods for receiving editing-related messages for the text view.

`protocol Theme`
Fonts and colors to be used by a TextView.

Structures

`struct DefaultTheme`
The theme used by a text view when no other theme has been specified.

`struct SearchQuery`
Query to search for in the text view.

`struct SearchResult`
A match returned by performing a search query.

Enumerations

`enum Language`
Language that can be used to syntax highlight text in the text view.

Documentation

Documentation.md

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as
- Adjust the height of lines.

- ! [Hero image showing a happy stone in
- ## ## Topics
- ### ### Essentials
- ``TextView``
 - ``TextViewDelegate``
- ### ### Syntax Highlighting
- ``Language``
 - ``Theme``
 - ``DefaultTheme``
- ### ### Find and Replace
- ``SearchQuery``
 - ``SearchResult``

Control Window Help

Search documentation

Tinystone

Essentials

```
class TextView
```

A type similar to UITextView with features commonly found in code editors.

```
protocol TextViewDelegate
```

The methods for receiving editing-related messages for the text view.

Syntax Highlighting

```
enum Language
```

Language that can be used to syntax highlight text in the text view.

```
protocol Theme
```

Fonts and colors to be used by a TextView.

```
struct DefaultTheme
```

The theme used by a text view when no other theme has been specified.

Find and Replace

```
struct SearchQuery
```

Query to search for in the text view.

```
struct SearchResult
```

A match returned by performing a search query.

Documentation.md

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as
- Adjust the height of lines.

![Hero image showing a happy stone in

Topics

Essentials

- ``TextView``
- ``TextViewDelegate``

Syntax Highlighting

- ``Language``
- ``Theme``
- ``DefaultTheme``

Find and Replace

- ``SearchQuery``
- ``SearchResult``

Framework

Tinystone

Performant code editor with syntax highlighting, line numbers, and invisible characters.



Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.



Topics

Essentials

- ↳ [Meet Tinystone](#)
Explore Tinystone, the performant code editor with syntax highlighting, line numbers, and invisible characters through a series of interactive tutorials.

Class

`class TextView`

Protocol

`protocol TextViewDelegate`
The methods for receiving editing-related messages for the text view.

Syntax Highlighting

`enum Language`
Language that can be used to syntax highlight text in the text view.

`protocol Theme`
Fonts and colors to be used by a TextView.

`struct DefaultTheme`
The theme used by a text view when no other theme has been specified.

Find and Replace

`struct SearchQuery`
Query to search for in the text view.

`struct SearchResult`
A match returned by performing a search query.

Landing Page

Class

TextView

A type similar to UITextView with features commonly found in code editors.

Declaration

```
final class TextView
```

Overview

TextView is a performant implementation of a text view with features such as showing line numbers, searching for text and replacing results, syntax highlighting, showing invisible characters and more.

To enable syntax highlighting in the text view, you must provide both a [Language](#) and a [Theme](#).

Topics

Essentials

```
init(frame: CGRect)
Create a new text view.
```

```
var text: String
The text that the text view displays.
```

```
var textViewDelegate: TextViewDelegate?
The object to receive editing callbacks.
```

```
protocol TextViewDelegate
The methods for receiving editing-related messages for the text view.
```

Text Layout

```
var isLineWrappingEnabled: Bool
When line wrapping is disabled, users can scroll the text view horizontally to see the entire line.
```

```
var showInvisibleCharacters: Bool
Enable to show invisible characters.
```

```
var lineHeightMultiplier: CGFloat
The line-height is multiplied with the value.
```

```
var kern: CGFloat
The number of points by which to adjust kern.
```

Syntax Highlighting

```
var language: Language
The language used for syntax highlighting text in the text view.
```

```
var theme: Theme
Fonts and colors used when syntax highlighting text.
```

Gutter

```
var showLineNumbers: Bool
Enable to show line numbers in the gutter.
```

```
var gutterWidth: CGFloat
Width of the gutter.
```

Extensions

Framework

Tinystone

Perfomant code editor with syntax highlighting, line numbers, and invisible characters.



Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features:

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.



Topics

Essentials

- ↳ [Meet Tinystone](#)
Explore Tinystone, the performant code editor with syntax highlighting, line numbers, and invisible characters through a series of interactive tutorials.

```
class TextView
    A type similar to UITextView with features commonly found in code editors.

protocol TextViewDelegate
    The methods for receiving editing-related messages for the text view.
```

Syntax Highlighting

```
enum Language
    Language that can be used to syntax highlight text in the text view.

protocol Theme
    Fonts and colors to be used by a TextView.

struct DefaultTheme
    The theme used by a text view when no other theme has been specified.
```

Find and Replace

```
struct SearchQuery
    Query to search for in the text view.

struct SearchResult
    A match returned by performing a search query.
```

Class

TextView

A type similar to UITextView with features commonly found in code editors.

Declaration

```
final class TextView
```

Overview

TextView is a performant implementation of a text view with features such as showing line numbers, searching for text and replacing results, syntax highlighting, showing invisible characters and more.

To enable syntax highlighting in the text view, you must provide both a [Language](#) and a [Theme](#).

Topics

Essentials

```
init(frame: CGRect)
    Create a new text view.

var text: String
    The text that the text view displays.

var textViewDelegate: TextViewDelegate?
    The object to receive editing callbacks.

protocol TextViewDelegate
    The methods for receiving editing-related messages for the text view.
```

Text Layout

```
var isLineWrappingEnabled: Bool
    When line wrapping is disabled, users can scroll the text view horizontally to see the entire line.

var showInvisibleCharacters: Bool
    Enable to show invisible characters.

var lineHeightMultiplier: CGFloat
    The line-height is multiplied with the value.

var kern: CGFloat
    The number of points by which to adjust kern.
```

Syntax Highlighting

```
var language: Language
    The language used for syntax highlighting text in the text view.

var theme: Theme
    Fonts and colors used when syntax highlighting text.
```

Gutter

```
var showLineNumbers: Bool
    Enable to show line numbers in the gutter.

var gutterWidth: CGFloat
    Width of the gutter.
```

Landing Page

Extensions

``Tinystone/TextView``

Topics

Essentials

- ``init(frame:)``
- ``text``
- ``textViewDelegate``
- ``TextviewDelegate``

Text Layout

- ``isLineWrappingEnabled``
- ``showInvisibleCharacters``
- ``lineHeightMultiplier``
- ``kern``

Syntax Highlighting

- ``language``
- ``theme``

Gutter

- ``showLineNumbers``
- ``gutterWidth``

Class

TextView

A type similar to UITextView with features commonly found in code editors.

Declaration

```
final class TextView
```

Overview

TextView is a performant implementation of a text view with features such as showing line numbers, searching for text and replacing results, syntax highlighting, showing invisible characters and more.

To enable syntax highlighting in the text view, you must provide both a [Language](#) and a [Theme](#).

Topics

Essentials

`init(frame: CGRect)`
Create a new text view.

`var text: String`
The text that the text view displays.

`var textViewDelegate: TextViewDelegate?`
The object to receive editing callbacks.

`protocol TextViewDelegate`
The methods for receiving editing-related messages for the text view.

Text Layout

`var isLineWrappingEnabled: Bool`
When line wrapping is disabled, users can scroll the text view horizontally to see the entire line.

`var showInvisibleCharacters: Bool`
Enable to show invisible characters.

`var lineHeightMultiplier: CGFloat`
The line-height is multiplied with the value.

`var kern: CGFloat`
The number of points by which to adjust kern.

Syntax Highlighting

`var language: Language`
The language used for syntax highlighting text in the text view.

`var theme: Theme`
Fonts and colors used when syntax highlighting text.

Gutter

`var showLineNumbers: Bool`
Enable to show line numbers in the gutter.

``Tinystone/TextView``

Topics

Essentials

- ``init(frame:)``
- ``text``
- ``textViewDelegate``
- ``TextviewDelegate``

Text Layout

- ``isLineWrappingEnabled``
- ``showInvisibleCharacters``
- ``lineHeightMultiplier``
- ``kern``

Syntax Highlighting

- ``language``
- ``theme``

Gutter

- ``showLineNumbers``
- ``gutterWidth``

Class

TextView

A type similar to UITextView with features commonly found in code editors.

Declaration

```
final class TextView
```

Overview

TextView is a performant implementation of a text view with features such as showing line numbers, searching for text and replacing results, syntax highlighting, showing invisible characters and more.

To enable syntax highlighting in the text view, you must provide both a [Language](#) and a [Theme](#).

Topics

Essentials

`init(frame: CGRect)`
Create a new text view.

`var text: String`
The text that the text view displays.

`var textViewDelegate: TextViewDelegate?`
The object to receive editing callbacks.

`protocol TextViewDelegate`
The methods for receiving editing-related messages for the text view.

Text Layout

`var isLineWrappingEnabled: Bool`
When line wrapping is disabled, users can scroll the text view horizontally to see the entire line.

`var showInvisibleCharacters: Bool`
Enable to show invisible characters.

`var lineHeightMultiplier: CGFloat`
The line-height is multiplied with the value.

`var kern: CGFloat`
The number of points by which to adjust kern.

Syntax Highlighting

`var language: Language`
The language used for syntax highlighting text in the text view.

`var theme: Theme`
Fonts and colors used when syntax highlighting text.

Gutter

`var showLineNumbers: Bool`
Enable to show line numbers in the gutter.

``Tinystone/TextView``

Topics

Essentials

- ``init(frame:)``
- ``text``
- ``textViewDelegate``
- ``TextViewDelegate``

Text Layout

- ``isLineWrappingEnabled``
- ``showInvisibleCharacters``
- ``lineHeightMultiplier``
- ``kern``

Syntax Highlighting

- ``language``
- ``theme``

Gutter

- ``showLineNumbers``
- ``gutterWidth``

Class

TextView

A type similar to UITextView with features commonly found in code editors.

Declaration

```
final class TextView
```

Overview

TextView is a performant implementation of a text view with features such as showing line numbers, searching for text and replacing results, syntax highlighting, showing invisible characters and more.

To enable syntax highlighting in the text view, you must provide both a [Language](#) and a [Theme](#).

Topics

Essentials

```
init(frame: CGRect)
```

Create a new text view.

```
var text: String
```

The text that the text view displays.

```
var textViewDelegate: TextViewDelegate?
```

The object to receive editing callbacks.

```
protocol TextViewDelegate
```

The methods for receiving editing-related messages for the text view.

Text Layout

```
var isLineWrappingEnabled: Bool
```

When line wrapping is disabled, users can scroll the text view horizontally to see the entire line.

```
var showInvisibleCharacters: Bool
```

Enable to show invisible characters.

```
var lineHeightMultiplier: CGFloat
```

The line-height is multiplied with the value.

```
var kern: CGFloat
```

The number of points by which to adjust kern.

Syntax Highlighting

```
var language: Language
```

The language used for syntax highlighting text in the text view.

```
var theme: Theme
```

Fonts and colors used when syntax highlighting text.

Gutter

```
var showLineNumbers: Bool
```

Enable to show line numbers in the gutter.



Describe Symbols



Add Structure



Add Supplemental
Content



Describe Symbols



Add Structure



Add Supplemental
Content

Article

Getting Started

This articles gets you started using Runestone in your project.

Installation

Runestone is distributed using the [Swift Package Manager](#). Install it in a project by adding it as a dependency in your Package.swift manifest or through "Package Dependencies" in project settings.

```
let package = Package(
    dependencies: [
        .package(url: "git@github.com:simonsb/runestone.git", from: "0.1.0")
    ]
)
```

Usage

The following steps describe how to add a `TextView` to an app and start syntax highlighting text.

- 1. Create a TextView**

The `TextView` is a subclass of `UIScrollView` and as such can be initialized like any other view. It has an API that is very similar to the one of `UITextView`.

```
let textView = TextView()
view.addSubview(textView)
```

The `textView` can be customized in a variety of ways. The following code snippet shows how to enable line numbers, show the selected line, show invisible characters and adjust the line-height. Refer to the documentation of `TextView` for a full overview of the settings.

```
// Show line numbers.
textView.showLineNumbers = true
// Highlight the selected line.
textView.lineSelectionDisplayType = .line
// Show a page guide after the 80th character.
textView.showPageGuide = true
textView.pageGuideColumn = 80
// Show invisible characters.
textView.showInvisibles = true
textView.showSpaces = true
textView.showBreaks = true
textView.showSoftLineBreaks = true
// Set the line-height to 136%
textView.lineHeightMultiplier = 1.3
```

- 2. Add a language**

To start using `TextView` in an app, you should add the language of the text you would like to syntax highlight. Runestone uses `Tree-sitter`, and as such, Tree-sitter languages to syntax highlight text. Runestone does not come with any languages by default. Refer to the [Adding a Tree-sitter Language](#) article for information on how to add a language to your app.

- 3. Add a theme**

The next step is to add a theme to your app. Runestone uses the `Theme` protocol to customize the appearance of a `TextView`. Refer to the [Creating a Theme](#) article for information on how to add a theme to your app.

- 4. Set the state of the text view**

After adding a language and a theme to your app, you are ready to set the language and theme on the text view.

You can set the language, theme, and text on your text view using the appropriate getters on an instance of `TextView`. However, setting the language or text is an expensive operation as it requires processing the visible part of the text. Therefore it is recommended that you initialize

TinyStone Essentials

Setting up a TextView

This tutorial guides you through creating a `TextView`, adding it to your view hierarchy, and configuring the `TextView`.

30min

Estimated Time

```
public final class TextView: UIScrollView {
    // Delegate to receive updates
    public weak var editorDelegate: UITextViewDelegate?
    // Whether the text view is in edit mode
    public private(set) var isEditing: Bool {
        didSet {
            if isEditing == oldValue {
                textView.isEditable = isEditing
            }
        }
    }
    public var text: String {
        get {
            return textView.text
        }
        set {
            textView.text = newValue
            contextSize = textView.contentSize
        }
    }
}
```

Section 1

Create a TextView

Create an instance of TinyStone's `TextView` and add it to the view hierarchy.

Step 1
Select `ViewController.swift` in the Project navigator.

Step 2
Add a title to the view controller.

Step 3
Create an instance of `TextView`.

Step 4
Add the text view to the view hierarchy.

Step 5

Articles

Tutorials

Getting Started

This article gets you started using Tinystone in your project.

Installation

Tinystone is distributed using the [Swift Package Manager] (<https://www.swift.org/package-manager/>). Install it in a project by adding it as a dependency in your `Package.swift` manifest or through "Package Dependencies" in project settings.

```
```swift
let package = Package(
 dependencies: [
 .package(url: "git@github.com:simonbs/Tinystone.git",
from: "0.1.0")
]
)
```
```

```

## ## Usage

### Article

## Getting Started

This article gets you started using Runestone in your project.

### Installation

Runestone is distributed using the [Swift Package Manager](#). Install it in a project by adding it as a dependency in your `Package.swift` manifest or through "Package Dependencies" in project settings.

```
let package = Package(
 dependencies: [
 .package(url: "git@github.com:simonbs/Runestone.git", from: "0.1.0")
]
)
```

### Usage

The following steps describe how to add a [TextView](#) to an app and start syntax highlighting text.

#### 1. Create a TextView

The [TextView](#) is a subclass of `UIScrollView` and as such can be initialized like any other view. It has an API that is very similar to the one of `UITextView`.

```
let textView = TextView()
view.addSubview(textView)
```

The text view can be customized in a variety of ways. The following code snippet shows how to enable line numbers, show the selected line, add a page guide, show invisible characters and adjust the line-height. Refer to the documentation of [TextView](#) for a full overview of the settings.

```
// Show line numbers.
textView.showLineNumbers = true
// Highlight the selected line.
textView.lineSelectionDisplayType = .line
// Show a page guide after the 80th character.
textView.showPageGuide = true
textView.pageGuideColumn = 80
// Show all invisible characters.
textView.showTabs = true
textView.showSpaces = true
textView.showLineBreaks = true
textView.showSoftLineBreaks = true
// Set the line-height to 130%
textView.lineHeightMultiplier = 1.3
```

#### 2. Add a language

To start using [TextView](#) in an app, you should add the language of the text you would like to syntax highlight. Runestone uses [Tree-sitter](#), and as such, Tree-sitter languages to syntax highlight text. Runestone does not come with any languages by default. Refer to the [Adding a Tree-sitter Language](#) article for information on how to add a language to your app.

#### 3. Add a theme

The next step is to add a theme to your app. Runestone uses the [Theme](#) protocol to customize the appearance of a [TextView](#). Refer to the [Creating a Theme](#) article for information on how to add a theme to your app.

#### 4. Set the state of the text view

After adding a language and a theme to your app, you are ready to set the language and theme on the text view.

# # ``Tinystone``

Performant code editor with syntax highlighting, line numbers, and invisible characters.

## ## Overview

Tinystone provides a text view with support for...

## ## Topics

### ### Essentials

- ``TextView``
- ``TextViewDelegate``

### ### Syntax Highlighting

- ``Language``
- ``Theme``
- ``DefaultTheme``

### ### Find and Replace

## Topics

### Essentials

`class TextView`  
A type similar to UITextView with features commonly found in code editors.

`protocol TextViewDelegate`  
The methods for receiving editing-related messages for the text view.

### Syntax Highlighting

`enum Language`  
Language that can be used to syntax highlight text in the text view.

`protocol Theme`  
Fonts and colors to be used by a TextView.

`struct DefaultTheme`  
The theme used by a text view when no other theme has been specified.

### Find and Replace

`struct SearchQuery`  
Query to search for in the text view.

`struct SearchResult`  
A match returned by performing a search query.

# # ``Tinystone``

Performant code editor with syntax highlighting, line numbers, and invisible characters.

## ## Overview

Tinystone provides a text view with support for...

## ## Topics

### ### Essentials

- <doc:Getting-Started>
- ``TextView``
- ``TextViewDelegate``

### ### Syntax Highlighting

- ``Language``
- ``Theme``
- ``DefaultTheme``

### ### Find and Replace

## Topics

### Essentials

#### [Getting Started](#)

This articles gets you started using Tinystone in your project.

#### `class TextView`

A type similar to UITextView with features commonly found in code editors.

#### `protocol TextViewDelegate`

The methods for receiving editing-related messages for the text view.

### Syntax Highlighting

#### `enum Language`

Language that can be used to syntax highlight text in the text view.

#### `protocol Theme`

Fonts and colors to be used by a TextView.

#### `struct DefaultTheme`

The theme used by a text view when no other theme has been specified.

### Find and Replace

#### `struct SearchQuery`

Query to search for in the text view.

#### `struct SearchResult`

A match returned by performing a search query.

**Article**

## Getting Started

This articles gets you started using Runestone in your project.

### Installation

Runestone is distributed using the [Swift Package Manager](#). Install it in a project by adding it as a dependency in your Package.swift manifest or through "Package Dependencies" in project settings.

```
jet package = Package(
 dependencies: [
 .package(url: "git@github.com:simonsb/runestone.git", from: "0.1.0")
]
)
```

### Usage

The following steps describe how to add a `TextView` to an app and start syntax highlighting text.

- 1. Create a TextView**  
The `TextView` is a subclass of `UIScrollView` and as such can be initialized like any other view. It has an API that is very similar to the one of `UITextView`.

```
jet textView = TextView()
view.addSubview(textView)
```

The text view can be customized in a variety of ways. The following code snippet shows how to enable line numbers, show the selected line, show invisible characters and adjust the line-height. Refer to the documentation of `TextView` for a full overview of the settings.

```
// Show line numbers.
textView.showLineNumbers = true
// Highlight the selected line.
textView.lineSelectionDisplayType = .line
// Show a page guide after the 80th character.
textView.showPageGuide = true
textView.pageGuideColumn = 80
// Show invisible characters.
textView.showTab = true
textView.showSpaces = true
textView.showBreaks = true
textView.showSoftLineBreaks = true
// Set the line-height to 136%
textView.lineHeightMultiplier = 1.3
```

- 2. Add a language**  
To start using `TextView` in an app, you should add the language of the text you would like to syntax highlight. Runestone uses `Tree-sitter`, and as such, Tree-sitter languages to syntax highlight text. Runestone does not come with any languages by default. Refer to the [Adding a Tree-sitter Language](#) article for information on how to add a language to your app.
- 3. Add a theme**  
The next step is to add a theme to your app. Runestone uses the `Theme` protocol to customize the appearance of a `TextView`. Refer to the [Creating a Theme](#) article for information on how to add a theme to your app.
- 4. Set the state of the text view**  
After adding a language and a theme to your app, you are ready to set the language and theme on the text view.

You can set the language, theme, and text on your text view using the appropriate getters on an instance of `TextView`. However, setting the language or text is an expensive operation as it requires processing the visible part of the text. Therefore it is recommended that you initialize

**TinyStone Essentials**

## Setting up a TextView

This tutorial guides you through creating a `TextView`, adding it to your view hierarchy, and configuring the `TextView`.

**30min**

Estimated Time

```
public final class TextView: UIScrollView {
 // ... (implementation details)
 public weak var editorDelegate: EditorDelegate?
 // Whether the text view is editing or not.
 public private(set) var isEditing: Bool {
 didSet {
 if isEditing == oldValue {
 textView.isEditing = oldValue
 }
 }
 }
 public var text: String {
 get {
 return textView.text
 }
 set {
 textView.text = newValue
 context.invalidateTextLayout()
 }
 }
}
```

### Section 1

#### Create a TextView

Create an instance of TinyStone's `TextView` and add it to the view hierarchy.

**Step 1**  
Select `ViewController.swift` in the Project navigator.

**Step 2**  
Add title to the view controller.

**Step 3**  
Create an instance of `TextView`.

**Step 4**  
Add the text view to the view hierarchy.

**Step 5**

## Articles

## Tutorials

Article

## Getting Started

This article gets you started using Runestone in your project.

### Installation

Runestone is distributed using the [Swift Package Manager](#). Install it in a project by adding it as a dependency in your Package.swift manifest or through "Package Dependencies" in project settings.

```
let package = Package(
 dependencies: [
 .package(url: "git@github.com:simonsb/Runestone.git", from: "0.1.0")
]
)
```

### Usage

The following steps describe how to add a `TextView` to an app and start syntax highlighting text.

- 1. Create a TextView**

The `TextView` is a subclass of `UIScrollView` and as such can be initialized like any other view. It has an API that is similar to the one of `UITextView`.

```
let textView = TextView()
view.addSubview(textView)
```

The `textView` can be customized in a variety of ways. The following code snippet shows how to enable line numbers, show the selected line, show invisible characters and adjust the line-height. Refer to the documentation of `TextView` for a full overview of the settings.

```
// Show line numbers.
textView.showLineNumbers = true
// Highlight the selected line.
textView.lineSelectionDisplayType = .line
// Show a page guide after the 80th character.
textView.showPageGuide = true
textView.pageGuideColumn = 80
// Show invisible characters.
textView.showInvisibles = true
textView.showSpaces = true
textView.showBreaks = true
textView.showSoftLineBreaks = true
// Set the lineheight to 130%
textView.lineHeightMultiplier = 1.3
```

- 2. Add a language**

To start using `TextView` in an app, you should add the language of the text you would like to syntax highlight. Runestone uses [Tree-sitter](#), and as such, Tree-sitter languages to syntax highlight text. Runestone does not come with any languages by default. Refer to the [Adding a Tree-sitter Language](#) article for information on how to add a language to your app.

- 3. Add a theme**

The next step is to add a theme to your app. Runestone uses the `Theme` protocol to customize the appearance of a `TextView`. Refer to the [Creating a Theme](#) article for information on how to add a theme to your app.

- 4. Set the state of the text view**

After adding a language and a theme to your app, you are ready to set the language and theme on the text view.

You can set the language, theme, and text on your text view using the appropriate getters on an instance of `TextView`. However, setting the language or text is an expensive operation as it requires processing the visible part of the text. Therefore it is recommended that you initialize

[https://github.com/simonsb/Runestone](#)  
 This repository is translated to Swift by [Tree-sitter](#) ([https://github.com/tree-sitter/tree-sitter](#)). Thanks to the effort of many contributors (at least), thanks!

## Tinystone Essentials

### Setting up a TextView

This tutorial guides you through creating a `TextView`, adding it to your view hierarchy, and configuring the `TextView`.

**30min**  
 Estimated Time

```
public final class TextView: UIScrollView {
 // Delegate to observe changes
 public weak var editorDelegate: TextViewEditorDelegate?
 // Whether the text view is in edit mode
 public private(set) var isEditing
 didSet {
 if isEditing == oldValue {
 textView.isEditable = isEditing
 }
 }
 public var text: String {
 get {
 return textView.text
 }
 set {
 textView.text = newValue
 contextSize = textView.contentSize
 }
 }
}
```

### Section 1

#### Create a TextView

Create an instance of Tinystone's `TextView` and add it to the view hierarchy.



**Step 1**  
 Select `ViewController.swift` in the Project navigator.

```
import UIKit
class ViewController: UIViewController {
 override func viewDidLoad() {
 super.viewDidLoad()
 title = "TextCompanion"
 }
}
```

**Step 2**  
 Add a title to the view controller.

**Step 3**  
 Create an instance of `TextView`.

**Step 4**  
 Add the `text` view to the view hierarchy.

**Step 5**



## Articles

## Tutorials

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

Swift

Featured

Workspace Documentation

Tinystone

- > Tinystone
- > Accessibility
- > ActivityKit
- > App Clips
- > AppKit
- > Bundle Resources
- > Core Foundation
- > Distributed
- > Foundation
- > MatterSupport
- > Objective-C Runtime
- > RegexBuilder
- > ScreenCaptureKit
- > Swift
- > SwiftUI
- > TVML
- > TVMLKit
- > TVUIKit
- > UIKit
- > WatchKit
- > watchOS apps

App Services

- > Accounts
- > Address Book
- > Address Book UI
- > AdServices
- > AdSupport
- > App Intents
- > Application Services

Filter

Tinystone > Tinystone > Meet Tinystone

Search documentation

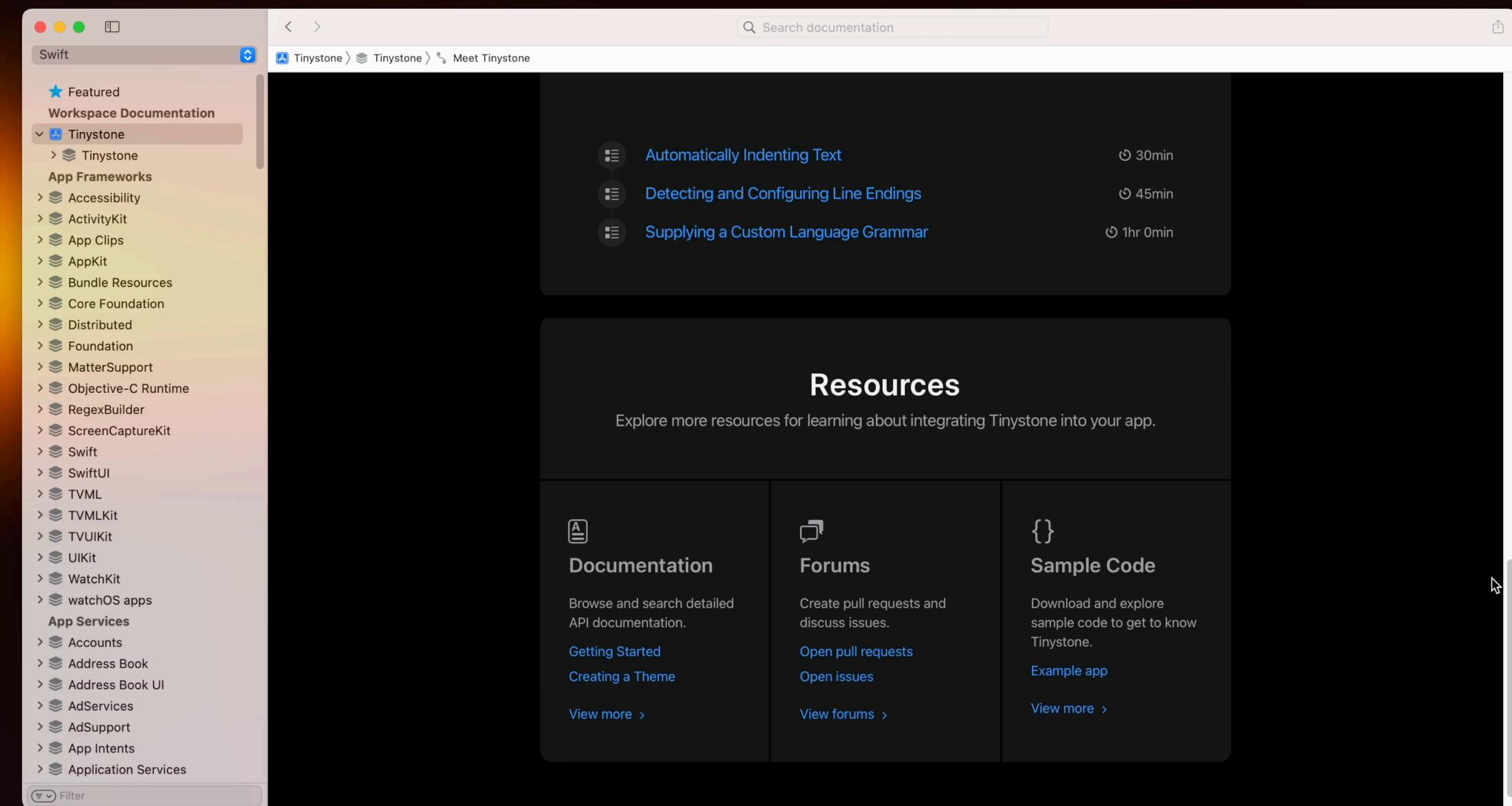
# Meet Tinystone

Explore Tinystone, the performant code editor with syntax highlighting, line numbers, and invisible characters through a series of interactive tutorials.



⌚ 3hr 5min Estimated Time

Get started



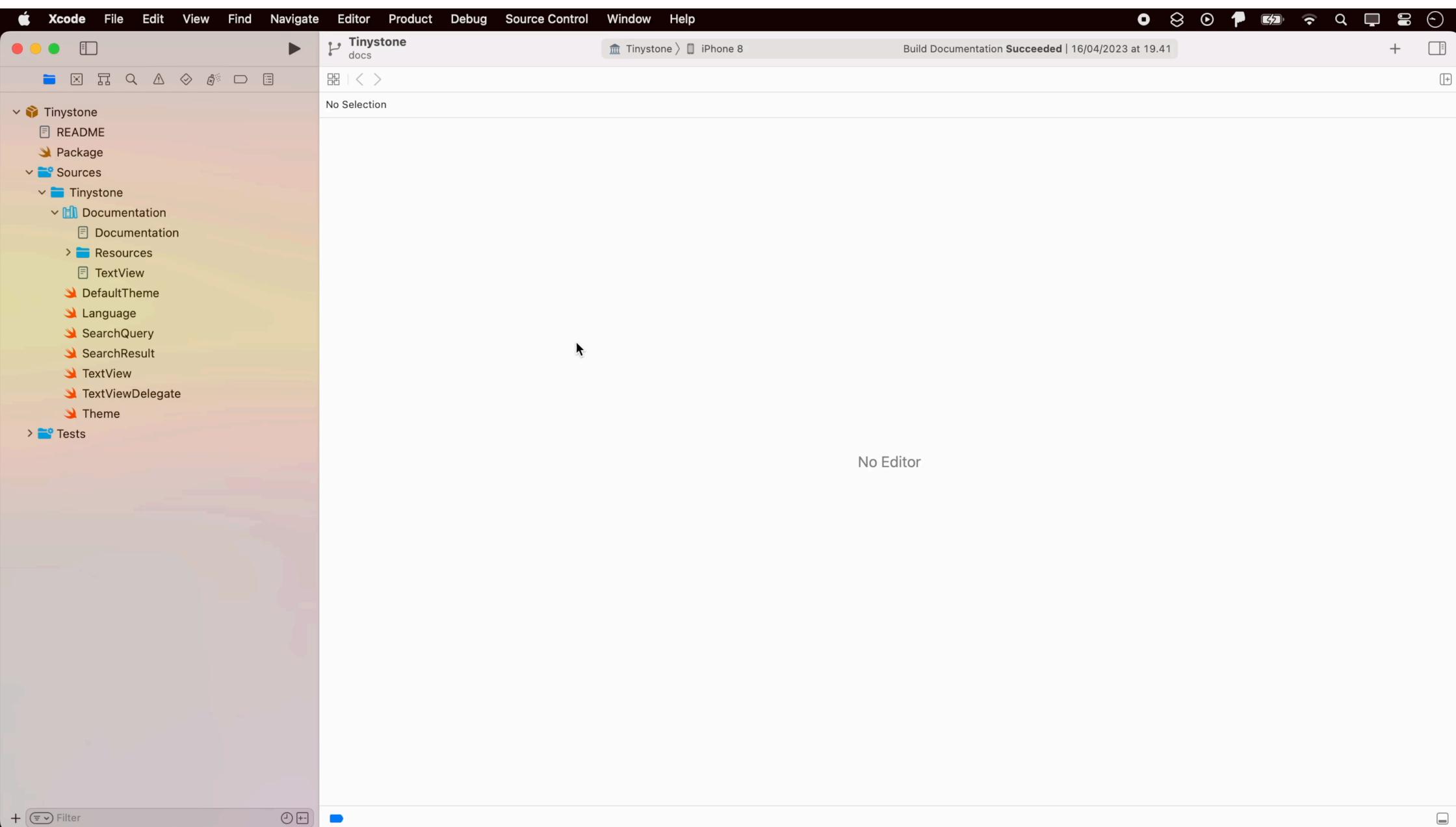
# DocC uses Markdown

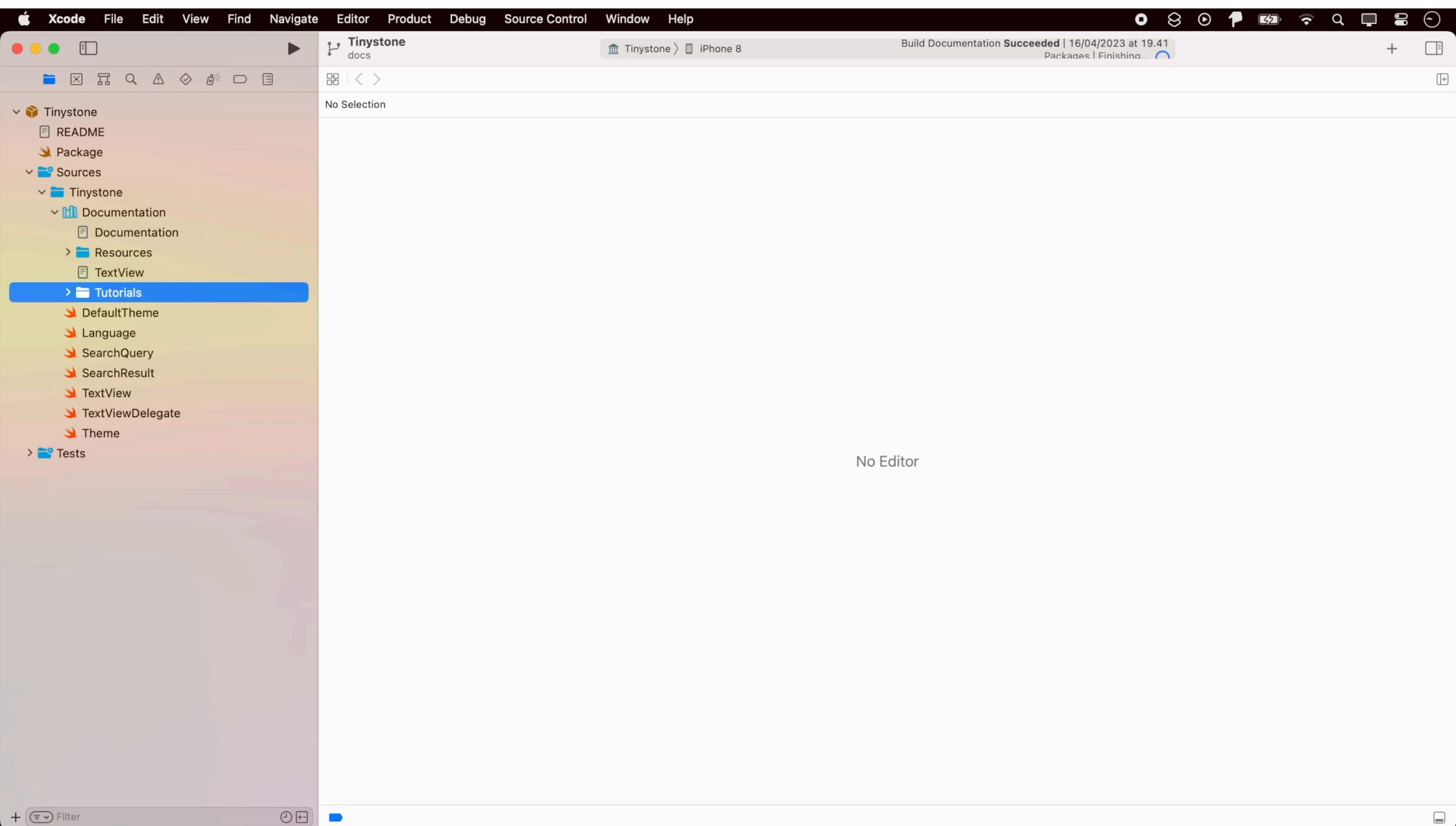
**DocC uses a dialect  
of Markdown**

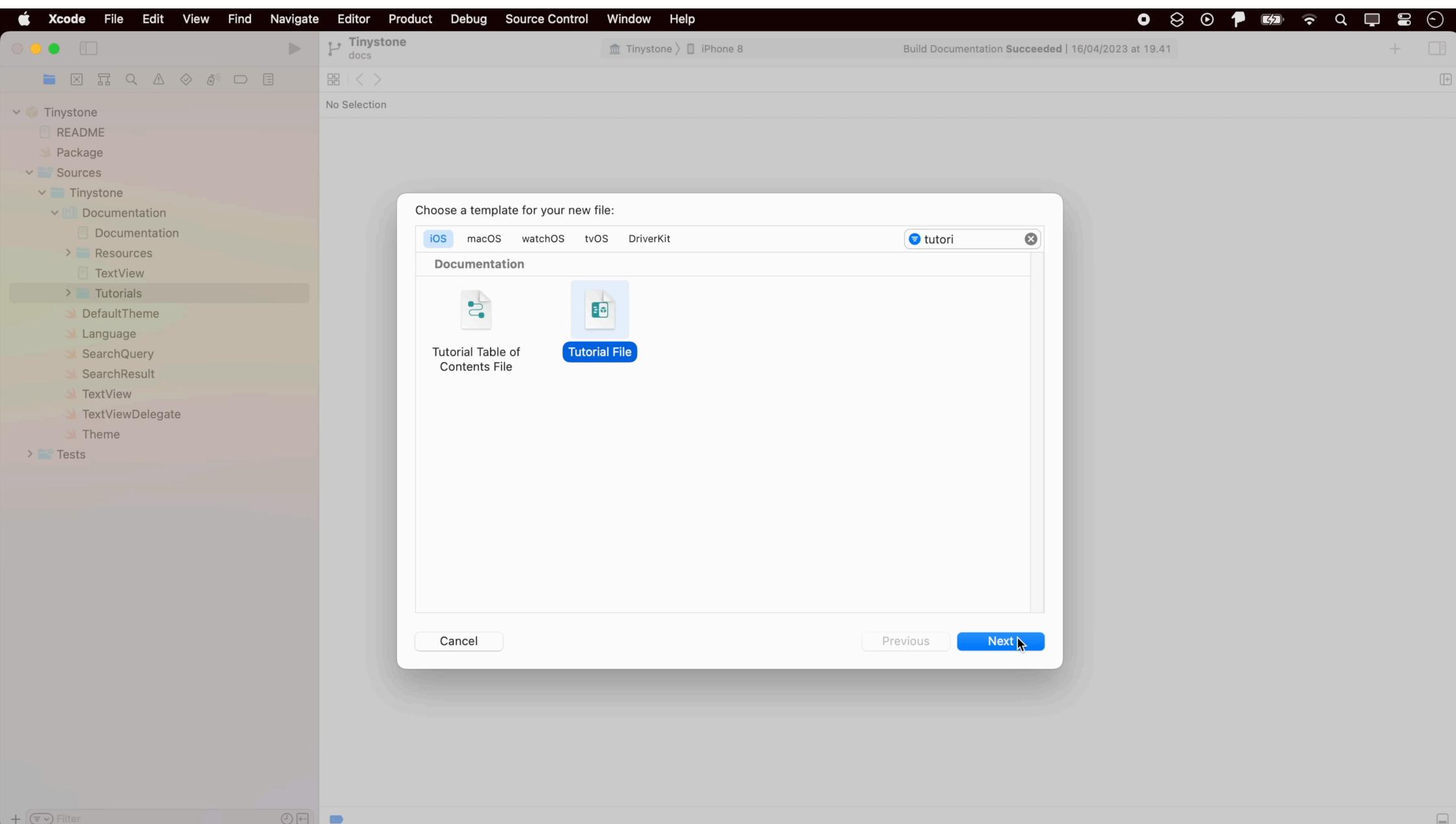
# Directives

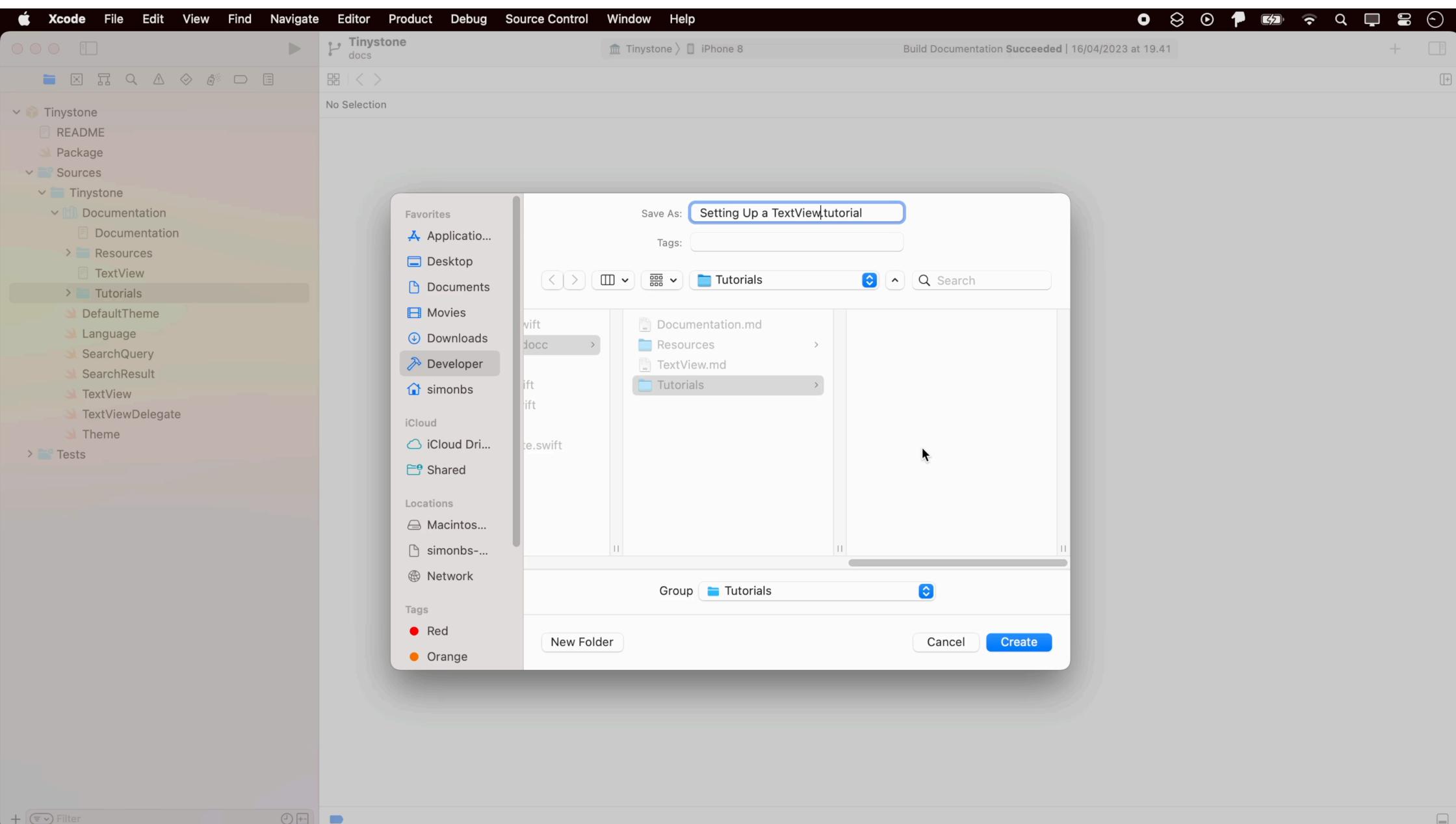
@Steps	@Intro	@Code	@Metadata
			@TechnologyRoot
	@Chapter		@Comment
@ContentAndMedia			
	@DisplayName		@TutorialReference
		@Tutorial	
@Image	@Section		@Step

```
@Code(name: name, file: file) {
 children
}
```









```
@Tutorial(time: number) {
 @Intro(title: "text") {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Section(title: "text") {
 @ContentAndMedia {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Steps {
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
 }
 }
}
```

## Setting Up a TextView.md

```
@Tutorial(time: number) {
 @Intro(title: "text") {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Section(title: "text") {
 @ContentAndMedia {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Steps {
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
 }
 }
}
```

### Tinystone Essentials

<#text#>

<#text#>

### Section 1

<#text#>

<#text#>

Step 1  
<#text#>

 Setting Up a TextView.md

```
@Tutorial(time: 30) {
 @Intro(title: "text") {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Section(title: "text") {
 @ContentAndMedia {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Steps {
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
 }
 }
}
```

Tinystone Essentials

<#text#>

<#text#>

30min

Estimated Time

Section 1

<#text#>

<#text#>

Step 1

<#text#>



```
@Tutorial(time: 30) {
 @Intro(title: "Setting Up a TextView") {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Section(title: "text") {
 @ContentAndMedia {
 text

 @Image(source: file, alt: "accessible description")
 }
 }

 @Steps {
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
 }
}
```

Tinystone Essentials

## Setting Up a TextView

<#text#>

30min

Estimated Time

Section 1

<#text#>

<#text#>

Step 1

<#text#>

## Setting Up a TextView.md

```
@Tutorial(time: 30) {
 @Intro(title: "Setting Up a TextView") {
 This tutorial guides you through creating a TextView and
 configuring it.

 @Image(source: file, alt: "accessible description")
 }

 @Section(title: "text") {
 @ContentAndMedia {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Steps {
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Step {
 text
 }
 }
 }
}
```

Tinystone Essentials

## Setting Up a TextView

This tutorial guides you through creating a TextView  
and configuring it.

30min

Estimated Time

Section 1

<#text#>

<#text#>

Step 1

<#text#>

## Setting Up a TextView.md

```
@Tutorial(time: 30) {
 @Intro(title: "Setting Up a TextView") {
 This tutorial guides you through creating a TextView and
 configuring it.

 @Image(source: header.png, alt: "Code snippets in grid")
 }

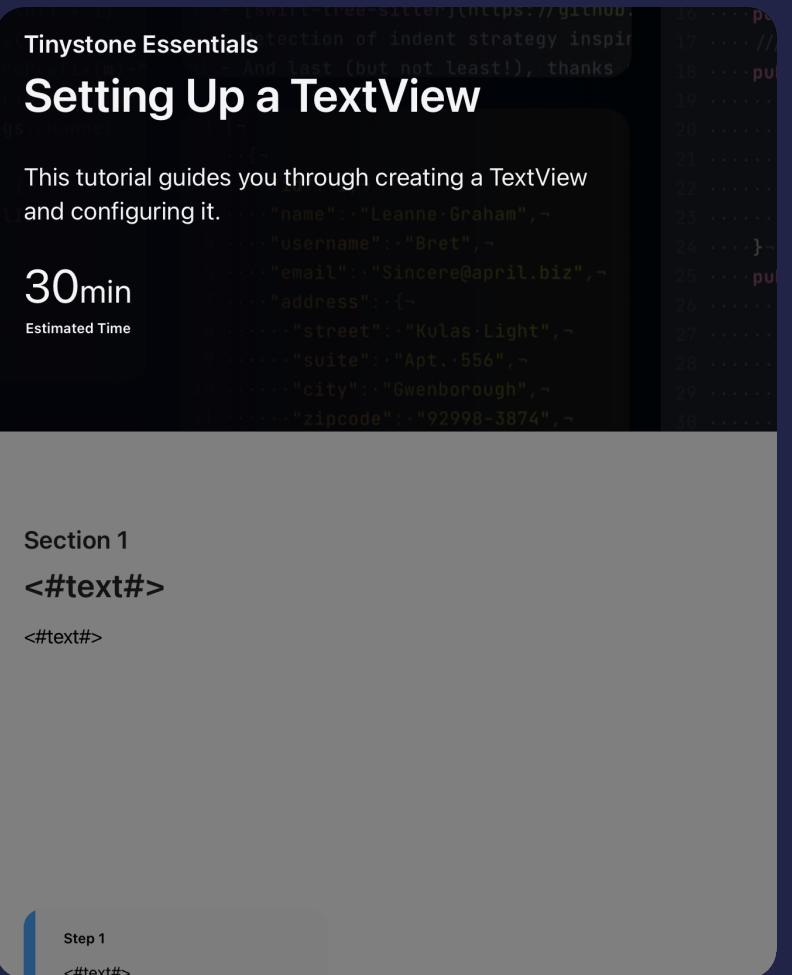
 @Section(title: "text") {
 @ContentAndMedia {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Steps {
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Step {
 text
 }
 }
 }
}
```



The screenshot shows a mobile application interface with a dark theme. At the top, there is a navigation bar with a back arrow and the text "Tinystone Essentials". Below the navigation bar is a large, semi-transparent overlay box containing the following information:

- Tinystone Essentials**
- Setting Up a TextView**
- This tutorial guides you through creating a TextView and configuring it.
- 30min** (Estimated Time)
- Section 1**
- <#text#>**
- Step 1**
- <#text#>**

The background of the app shows some code snippets and a list of names.

## Setting Up a TextView.md

```
@Tutorial(time: 30) {
 @Intro(title: "Setting Up a TextView") {
 This tutorial guides you through creating a TextView and
 configuring it.

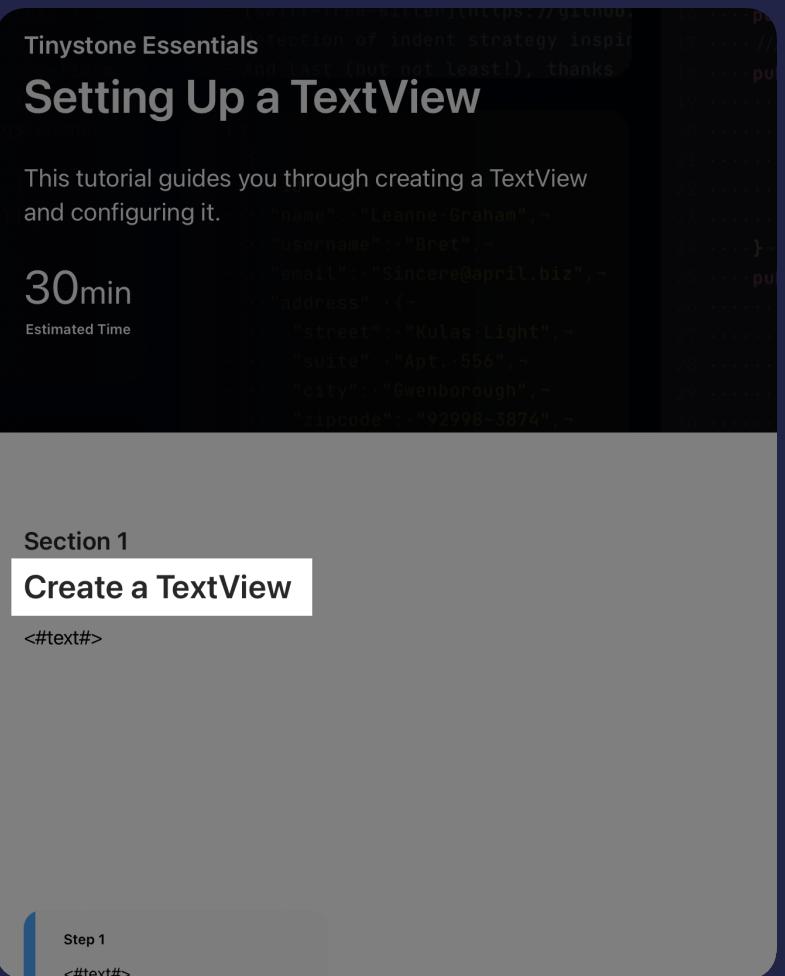
 @Image(source: header.png, alt: "Code snippets in grid")
 }

 @Section(title: "Create a TextView") {
 @ContentAndMedia {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Steps {
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
 }
 }
}
```



The screenshot shows a mobile application interface. At the top, there's a dark header with the text "Tinystone Essentials" and a subtitle "Setting Up a TextView". Below the header, the main content area has a dark background with white text. It starts with a paragraph: "This tutorial guides you through creating a TextView and configuring it." To the right of this text is some sample JSON-like code. Below the paragraph is a large button labeled "30min" which says "Estimated Time" underneath. The main content area is divided into sections. The first section is titled "Section 1" and contains a sub-section titled "Create a TextView". Inside this section, there's a placeholder text "`<#text#>`". At the bottom of the main content area, there's a small callout box labeled "Step 1" with the same placeholder text "`<#text#>`".

## Setting Up a TextView.md

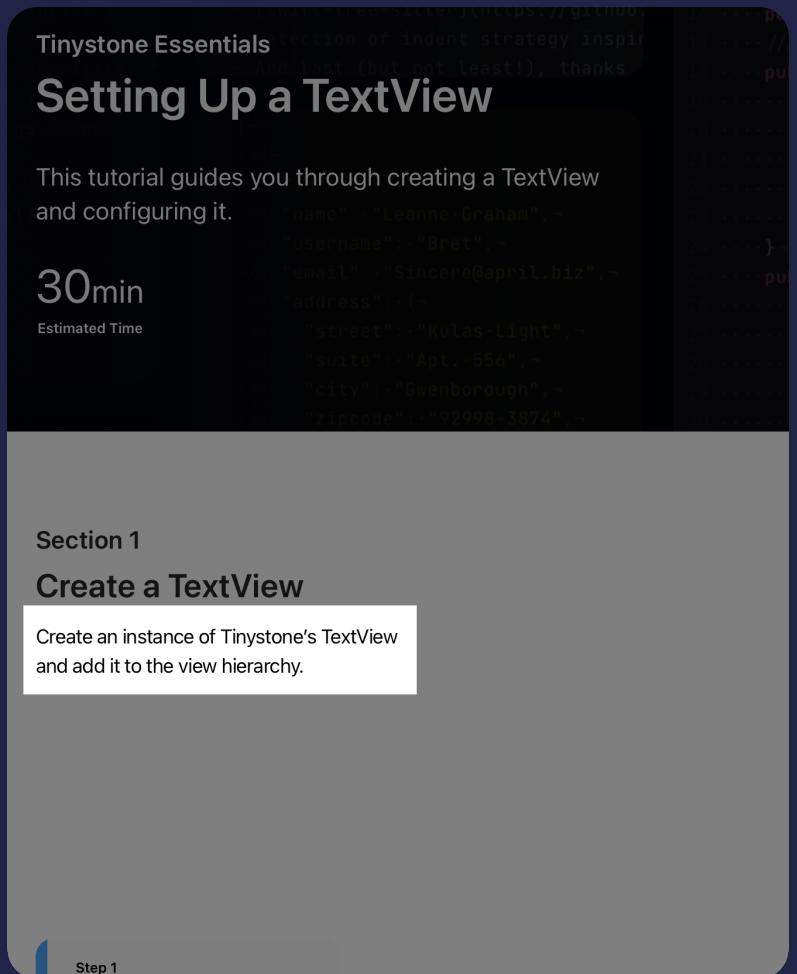
```
@Tutorial(time: 30) {
 @Intro(title: "Setting Up a TextView") {
 This tutorial guides you through creating a TextView and
 configuring it.

 @Image(source: header.png, alt: "Code snippets in grid")
 }

 @Section(title: "Create a TextView") {
 @ContentAndMedia {
 Create an instance of Tinystone's TextView and add it to
 the view hierarchy.

 @Image(source: file, alt: "accessible description")
 }

 @Steps {
 @Step {
 text
 @Image(source: file, alt: "accessible description")
 }
 @Step {
 text
 }
 }
 }
}
```



The screenshot shows a mobile application interface with a dark background. At the top, there is a navigation bar with a back arrow and a search icon. Below the navigation bar, there is a header section with a profile picture and the text "Tinystone Essentials". The main content area features a large title "Setting Up a TextView" in bold. Below the title, there is a brief description: "This tutorial guides you through creating a TextView and configuring it." To the left of the description, there is a timer icon showing "30min" and the text "Estimated Time". On the right side of the description, there is some sample JSON code. At the bottom of the screen, there is a large button labeled "Section 1" and "Create a TextView" with a sub-instruction: "Create an instance of Tinystone's TextView and add it to the view hierarchy." A small blue progress bar at the bottom indicates the step number "Step 1".

## Setting Up a TextView.md

```
@Tutorial(time: 30) {
 @Intro(title: "Setting Up a TextView") {
 This tutorial guides you through creating a TextView and
 configuring it.

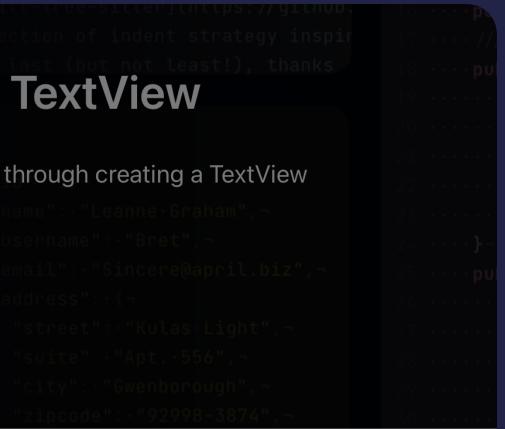
 @Image(source: header.png, alt: "Code snippets in grid")
 }

 @Section(title: "Create a TextView") {
 @ContentAndMedia {
 Create an instance of Tinystone's TextView and add it to
 the view hierarchy.

 @Image(source: section1.png, alt: "Xcode on a MacBook")
 }

 @Steps {
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
 @Step {
 text
 }
 }
 }
}
```



**Tinystone Essentials**

## Setting Up a TextView

This tutorial guides you through creating a TextView and configuring it.

**30min**  
Estimated Time

**Section 1**

### Create a TextView

Create an instance of Tinystone's TextView and add it to the view hierarchy.

Step 1



## Setting Up a TextView.md

```
@Tutorial(time: 30) {
 @Intro(title: "Setting Up a TextView") {
 This tutorial guides you through creating a TextView and
 configuring it.

 @Image(source: header.png, alt: "Code snippets in grid")
 }

 @Section(title: "Create a TextView") {
 @ContentAndMedia {
 Create an instance of Tinystone's TextView and add it to
 the view hierarchy.

 @Image(source: section1.png, alt: "Xcode on a MacBook")
 }

 @Steps {
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
 @Step {
 text
 }
 }
 }
}
```

Tinystone Essentials

# Setting Up a TextView

This tutorial guides you through creating a TextView and configuring it.

30min

Estimated Time

Section 1

## Create a TextView

Create an instance of Tinystone's TextView and add it to the view hierarchy.



Step 1



## Setting Up a TextView.md

Create an instance of Tinystone's TextView and add it to the view hierarchy.

```
@Image(source: section1.png, alt: "Xcode on a MacBook")
}

@Steps {
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
}
```

### Section 1

#### Create a TextView

Create an instance of Tinystone's TextView and add it to the view hierarchy.



##### Step 1

<#text#>

##### Step 2

<#text#>



## Setting Up a TextView.md

Create an instance of Tinystone's TextView and add it to the view hierarchy.

```
@Image(source: section1.png, alt: "Xcode on a MacBook")
}

@Steps {
 @Step {
 Select ViewController.swift in the Project navigator.

 @Image(source: file, alt: "accessible description")
 }

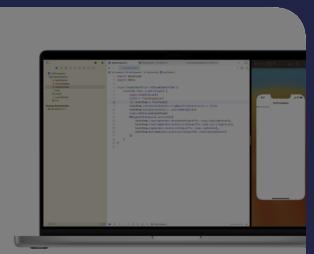
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
}
```

### Section 1

#### Create a TextView

Create an instance of Tinystone's TextView and add it to the view hierarchy.



##### Step 1

Select ViewController.swift in the Project navigator.

##### Step 2

<#text#>



## Setting Up a TextView.md

Create an instance of Tinystone's TextView and add it to the view hierarchy.

```
@Image(source: section1.png, alt: "Xcode on a MacBook")
}

@Steps {
 @Step {
 Select ViewController.swift in the Project navigator.

 @Code(name: "ViewContr.swift", file: code-1.swift) {

 }
 }

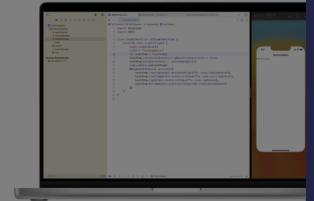
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
}
}
```

### Section 1

#### Create a TextView

Create an instance of Tinystone's TextView and add it to the view hierarchy.



##### Step 1

Select ViewController.swift in the Project navigator.

ViewController.swift

```
1 import UIKit
2
3 class ViewController: UIViewController {
4 override func viewDidLoad() {
5 super.viewDidLoad()
6 // Do any additional setup after loading the view.
7 }
8 }
```

##### Step 2

<#text#>

The screenshot shows the Xcode interface with a project named "Tinystone". The left sidebar displays the project structure, including a "Documentation" folder containing "DefaultTheme", "Language", "SearchQuery", "SearchResult", "TextView", "TextViewDelegate", and "Theme". The "Code" folder contains "code-1" and "code-2", with "code-1" currently selected. The main editor area shows the following Swift code:

```
1 import UIKit
2
3 class ViewController: UIViewController {
4 override func viewDidLoad() {
5 super.viewDidLoad()
6 // Do any additional setup after loading the view.
7 }
8 }
```



## Setting Up a TextView.md

Create an instance of Tinystone's TextView and add it to the view hierarchy.

```
@Image(source: section1.png, alt: "Xcode on a MacBook")
}

@Steps {
 @Step {
 Select ViewController.swift in the Project navigator.

 @Code(name: "ViewContr.swift", file: code-1.swift) {

 }
 }

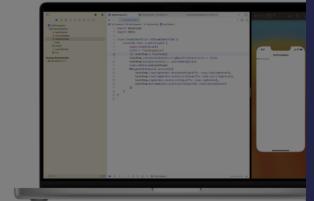
 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
}
}
```

### Section 1

#### Create a TextView

Create an instance of Tinystone's TextView and add it to the view hierarchy.



##### Step 1

Select ViewController.swift in the Project navigator.

ViewController.swift

```
1 import UIKit
2
3 class ViewController: UIViewController {
4 override func viewDidLoad() {
5 super.viewDidLoad()
6 // Do any additional setup after loading the view.
7 }
8 }
```

##### Step 2

<#text#>



## Setting Up a TextView.md

Create an instance of `tinystone's` `textView` and add it to the view hierarchy.

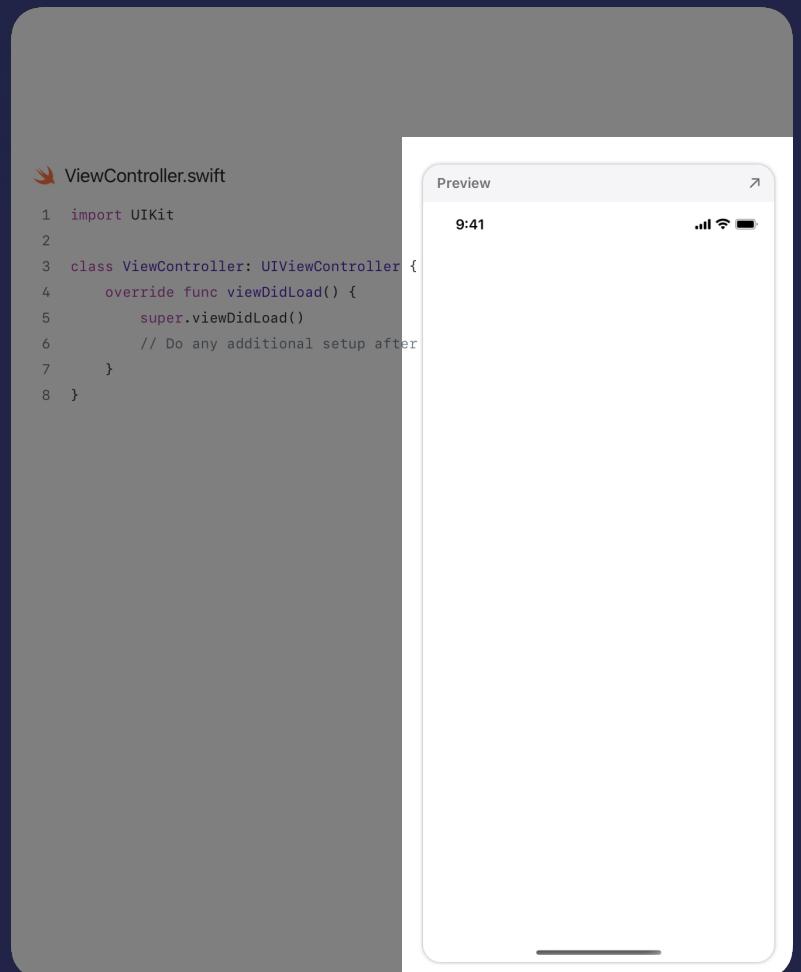
```
@Image(source: section1.png, alt: "Xcode on a MacBook")
}

@Steps {
 @Step {
 Select ViewController.swift in the Project navigator.

 @Code(name: "ViewContr.swift", file: code-1.swift) {
 @Image(source: preview.png, "Blank screen")
 }
 }

 @Step {
 text

 @Image(source: file, alt: "accessible description")
 }
}
```





## Setting Up a TextView.md

Create an instance of UITextView and add it to the view hierarchy.

```
@Image(source: section1.png, alt: "Xcode on a MacBook")
}

@Steps {
 @Step {
 Select ViewController.swift in the Project navigator.

 @Code(name: "ViewContr.swift", file: code-1.swift) {
 @Image(source: preview.png, "Blank screen")
 }
 }

 @Step {
 Add an instance of TextView to the view.

 @Code(name: "ViewContr.swift", file: code-2.swift) {
 @Image(source: preview-2.png, "Sample text view")
 }
 }
}
```

**Step 1**  
Select ViewController.swift in the Project navigator.

**Step 2**  
Add an instance of TextView to the view.

```
ViewController.swift
1 import UIKit
2
3 class ViewController: UIViewController {
4 override func viewDidLoad() {
5 super.viewDidLoad()
6 let textView = UITextView()
7 textView.translatesAutoresizingMaskIntoConstraints = false
8 view.addSubview(textView)
9 NSLayoutConstraint.activate([
10 textView.leadingAnchor.constraint(equalTo: view.leadingAnchor),
11 textView.trailingAnchor.constraint(equalTo: view.trailingAnchor),
12 textView.topAnchor.constraint(equalTo: view.topAnchor),
13 textView.bottomAnchor.constraint(equalTo: view.bottomAnchor)
14])
15 }
16 }
```



## Setting Up a TextView.md

Create an instance of TinyStone's TextView and add it to the view hierarchy.

```
@Image(source: section1.png, alt: "Xcode on a MacBook")
}

@Steps {
 @Step {
 Select ViewController.swift in the Project navigator.

 @Code(name: "ViewContr.swift", file: code-1.swift) {
 @Image(source: preview.png, "Blank screen")
 }
 }

 @Step {
 Add an instance of TextView to the view.

 @Code(name: "ViewContr.swift", file: code-2.swift) {
 @Image(source: preview-2.png, "Sample text view")
 }
 }
}
```

 **ViewController.swift**

```
1 import UIKit
2
3 class ViewController: UIViewController {
4 override func viewDidLoad() {
5 super.viewDidLoad()
6 let textView = TextView()
7 textView.translatesAutoresizingMaskIntoConstraints = false
8 view.addSubview(textView)
9 NSLayoutConstraint.activate([
10 textView.leadingAnchor.constraint(equalTo: view.leadingAnchor),
11 textView.trailingAnchor.constraint(equalTo: view.trailingAnchor),
12 textView.topAnchor.constraint(equalTo: view.topAnchor),
13 textView.bottomAnchor.constraint(equalTo: view.bottomAnchor)
14])
15 }
16 }
```

**Step 1**  
Select ViewController.swift in the Project navigator.

**Step 2**  
Add an instance of TextView and to the view.



## Setting Up a TextView.md

Create an instance of TinyStone's TextView and add it to the view hierarchy.

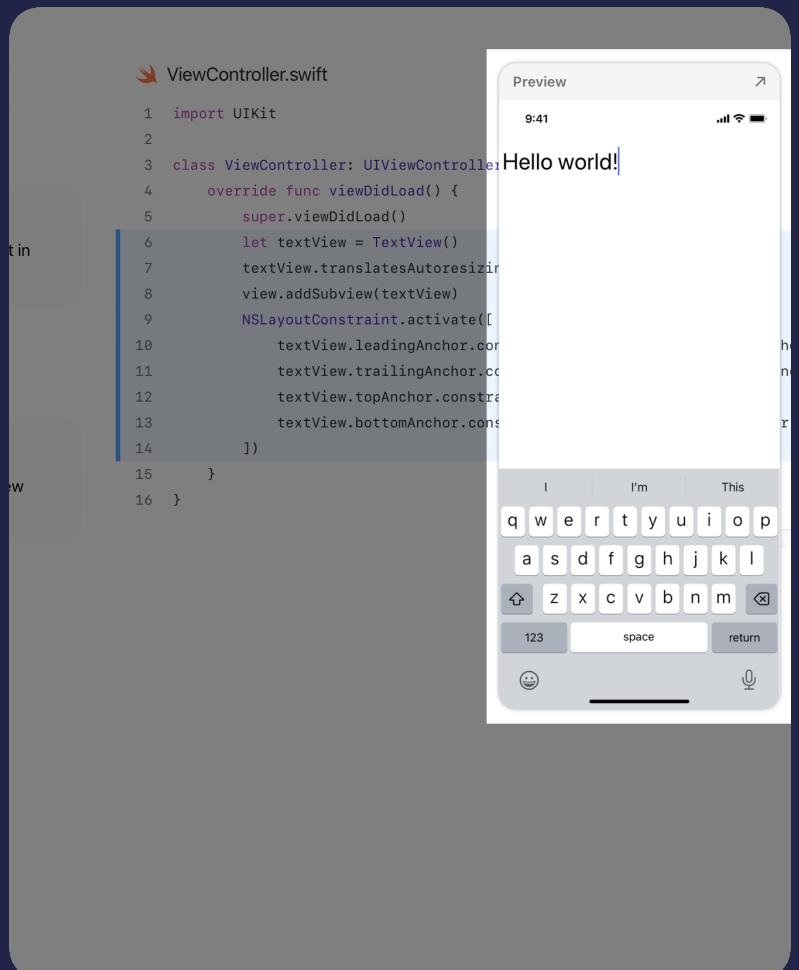
```
@Image(source: section1.png, alt: "Xcode on a MacBook")
}

@Steps {
 @Step {
 Select ViewController.swift in the Project navigator.

 @Code(name: "ViewContr.swift", file: code-1.swift) {
 @Image(source: preview.png, "Blank screen")
 }
 }

 @Step {
 Add an instance of TextView to the view.

 @Code(name: "ViewContr.swift", file: code-2.swift) {
 @Image(source: preview-2.png, "Sample text view")
 }
 }
}
```





Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

Tinystone main Tinystone iPhone 14 Pro Build Documentation Succeeded | Today at 12.23 1

Setting Up a TextView

Tinystone Sources Tinystone Documentation Tutorials Setting Up a TextView Image

1 @Tutorial(time: 30) { ! The tutorial 'Setting-Up-a-TextView' must be referenced from a Tu...  
2 @Intro(title: "Setting up a TextView") {  
3 This tutorial guides you through creating a TextView, adding it  
4 to your view hierarchy, and configuring the TextView.  
5 @Image(source: tutorial-header.png, alt: "Syntax highlighted  
code snippets in a grid.")  
6 }  
7  
8 @Section(title: "Create a TextView") {  
9 @ContentAndMedia {  
10 Create an instance of Tinystone's TextView and add it to  
11 the view hierarchy.  
12 @Image(source: getting-started-1.png, alt: "A MacBook Pro  
running Xcode.")  
13 }  
14  
15 @Steps {  
16 @Step {  
17 Select ViewController.swift in the Project navigator.  
18  
19 @Code(name: "ViewController.swift", file:  
getting-started-1-1.swift) {

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

Tinystone main Tinystone > iPhone 14 Pro

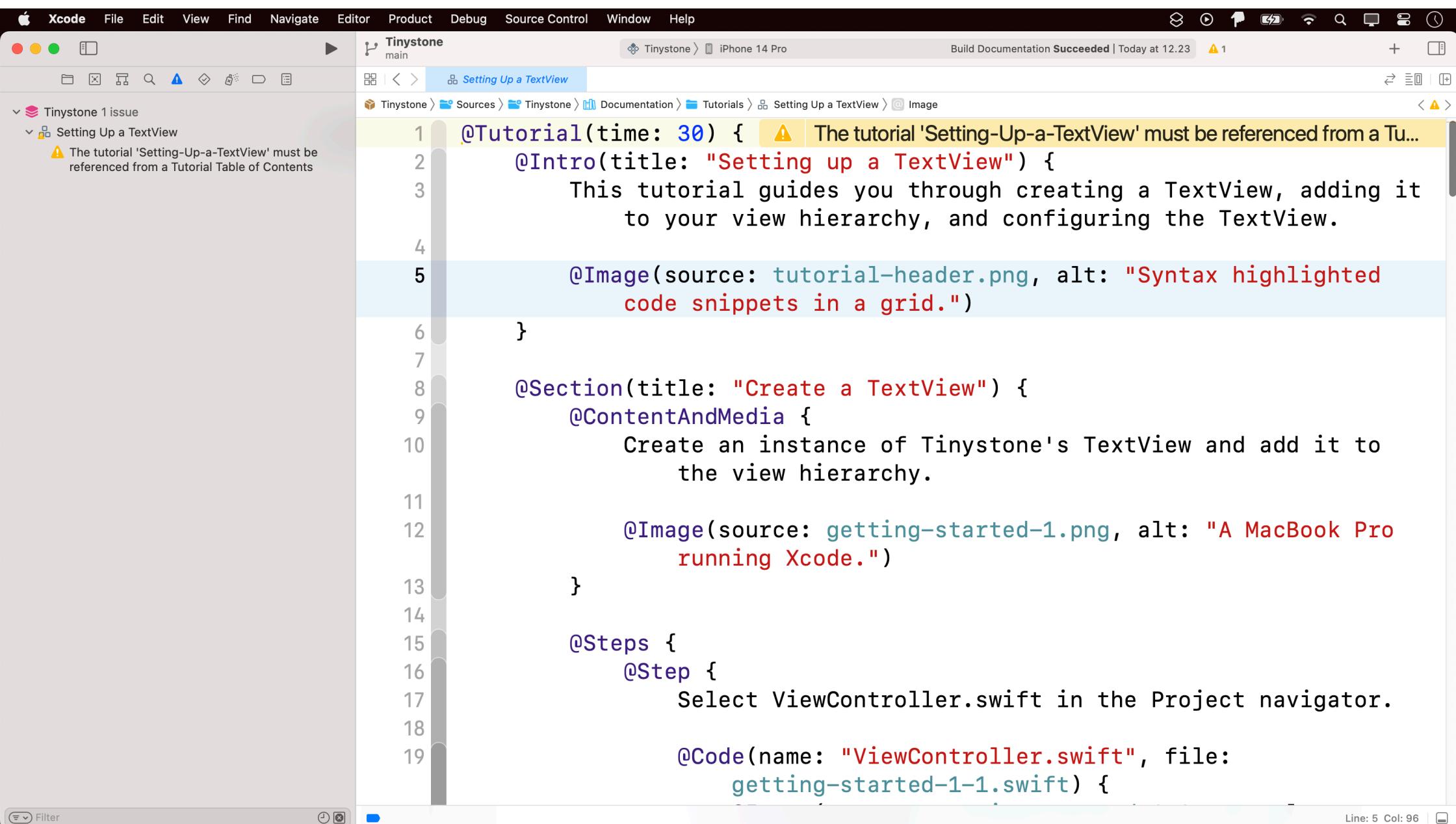
Setting Up a TextView

Tinystone 1 issue

Setting Up a TextView

The tutorial 'Setting-Up-a-TextView' must be referenced from a Tutorial Table of Contents

```
1 @Tutorial(time: 30) {
2 @Intro(title: "Setting
3 This tutorial guid
4 to your view h
5 @Image(source: tut
6 code snippets
7 }
8 @Section(title: "Create
9 @ContentAndMedia {
10 Create an inst
```



Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

Tinystone docs Tinystone > iPhone 14 Pro Build Documentation Succeeded | Today at 12.23 | 1

Setting Up a TextView

Tinystone > Sources > Tinystone > Documentation > Tutorials > Setting Up a TextView > Intro

1 @Tutorial(time: 30) { ⚠️ The tutorial 'Setting-Up-a-TextView' must be referenced from a Tu...

2     @Intro(title: "Setting up a TextView") {

3         This tutorial guides you through creating a TextView, adding it  
4         to your view hierarchy, and configuring the TextView.

5         @Image(source: tutorial-header.png, alt: "Syntax highlighted  
6             code snippets in a grid.")  
7     }

8     @Section(title: "Create a TextView") {

9         @ContentAndMedia {

10             Create an instance of Tinystone's TextView and add it to  
11             the view hierarchy.

12             @Image(source: getting-started-1.png, alt: "A MacBook Pro  
13                 running Xcode.")  
14     }

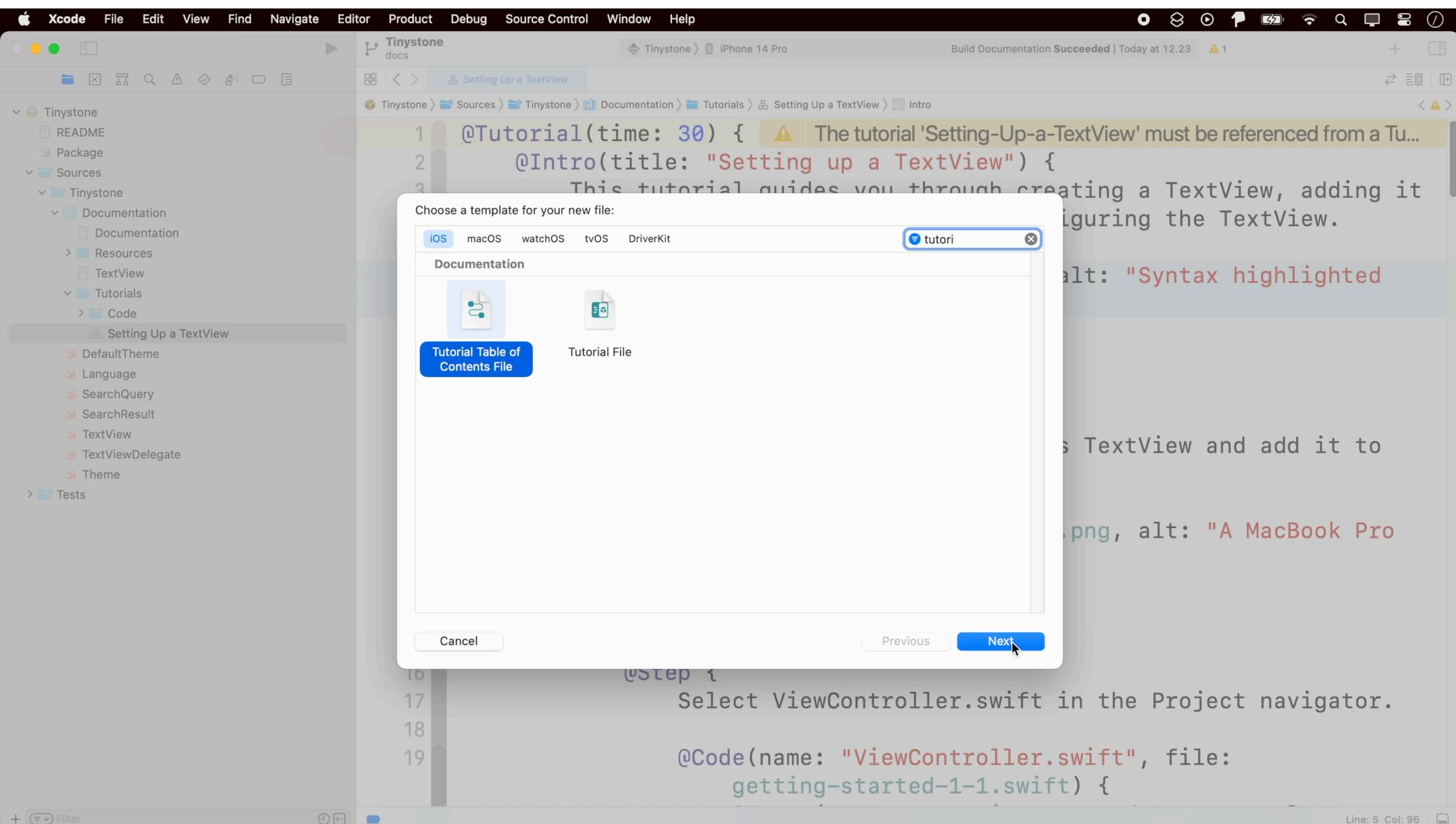
15     @Steps {

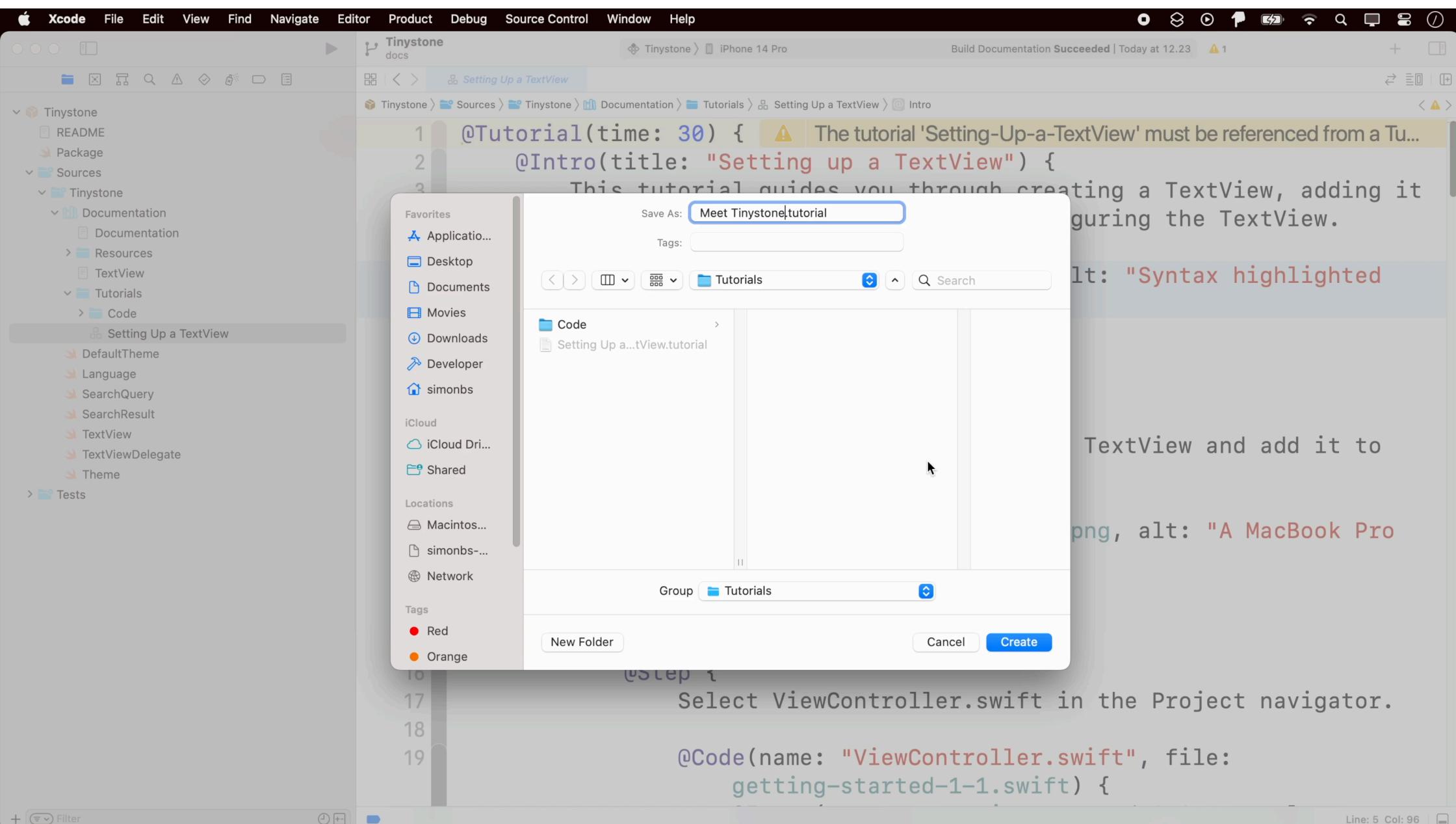
16         @Step {

17             Select ViewController.swift in the Project navigator.

18         @Code(name: "ViewController.swift", file:  
19             getting-started-1-1.swift) {

Line: 5 Col: 96





```
@Tutorials(name: "text") {
 @Intro(title: "text") {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Chapter(name: "text") {
 text

 @Image(source: file, alt: "accessible description")

 @TutorialReference(tutorial: "doc:tutorial name")
 }
}
```



```
@Tutorials(name: "text") {
 @Intro(title: "text") {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Chapter(name: "text") {
 text

 @Image(source: file, alt: "accessible description")

 @TutorialReference(tutorial: "doc:tutorial name")
 }
}
```

<#text#>

<#text#>



```
Meet Tinystone.md

@Tutorials(name: "Tinystone") {
 @Intro(title: "text") {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Chapter(name: "text") {
 text

 @Image(source: file, alt: "accessible description")

 @TutorialReference(tutorial: "doc:tutorial name")
 }
}
```

<#text#>

<#text#>



```
@Tutorials(name: "Tinystone") {
 @Intro(title: "Meet Tinystone") {
 text

 @Image(source: file, alt: "accessible description")
 }

 @Chapter(name: "text") {
 text

 @Image(source: file, alt: "accessible description")

 @TutorialReference(tutorial: "doc:tutorial name")
 }
}
```

## Meet Tinystone

<#text#>



```
@Tutorials(name: "Tinystone") {
 @Intro(title: "Meet Tinystone") {
 Explore Tinystone, the performant code editor,
 through a series of interactive tutorials.

 @Image(source: file, alt: "accessible description")
 }

 @Chapter(name: "text") {
 text

 @Image(source: file, alt: "accessible description")

 @TutorialReference(tutorial: "doc:tutorial name")
 }
}
```

The screenshot shows a dark-themed documentation page. At the top, there's a header section with the title 'Meet Tinystone' in a large, bold, light-colored font. Below the title is a dark gray box containing the text 'Explore Tinystone, the performant code editor, through a series of interactive tutorials.' In the main content area, there is a large, empty gray rectangular space, likely a placeholder for an image or a detailed description.



```
@Tutorials(name: "Tinystone") {
 @Intro(title: "Meet Tinystone") {
 Explore Tinystone, the performant code editor,
 through a series of interactive tutorials.

 @Image(source: hero.png, alt: "Happy stone in many colors")
 }

 @Chapter(name: "text") {
 text

 @Image(source: file, alt: "accessible description")

 @TutorialReference(tutorial: "doc:tutorial name")
 }
}
```

## Meet Tinystone

Explore Tinystone, the performant code editor, through a series of interactive tutorials.





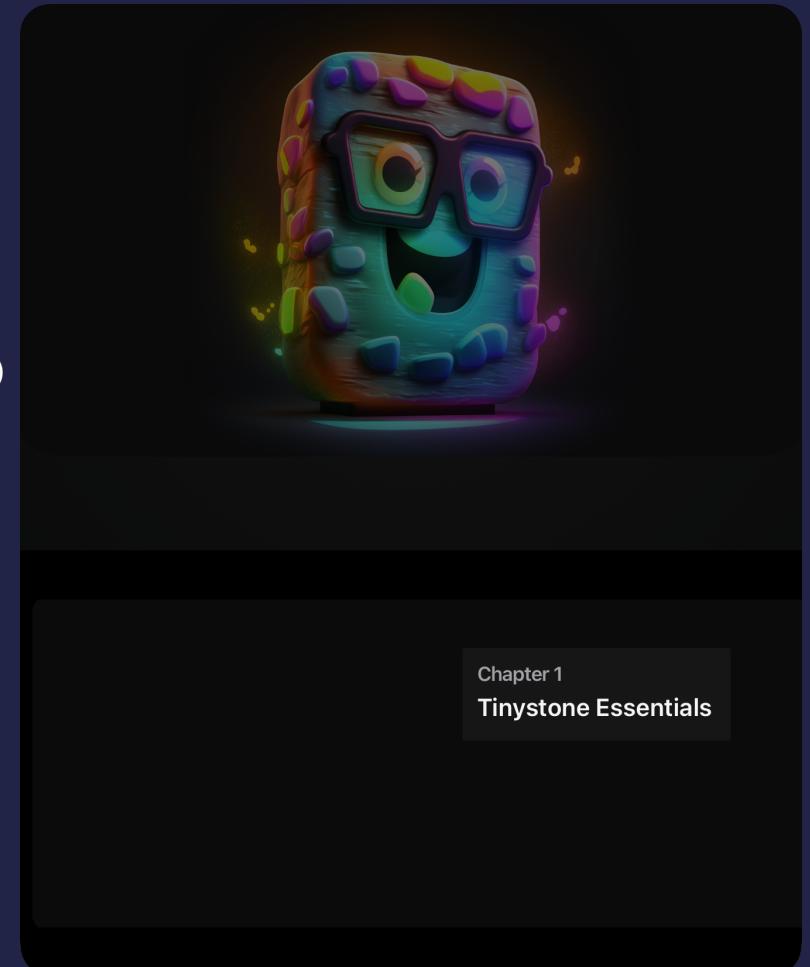
```
@Tutorials(name: "Tinystone") {
 @Intro(title: "Meet Tinystone") {
 Explore Tinystone, the performant code editor,
 through a series of interactive tutorials.

 @Image(source: hero.png, alt: "Happy stone in many colors")
 }
}
```

```
@Chapter(name: "Tinystone Essentials") {
 text

 @Image(source: file, alt: "accessible description")

 @TutorialReference(tutorial: "doc:tutorial name")
}
}
```





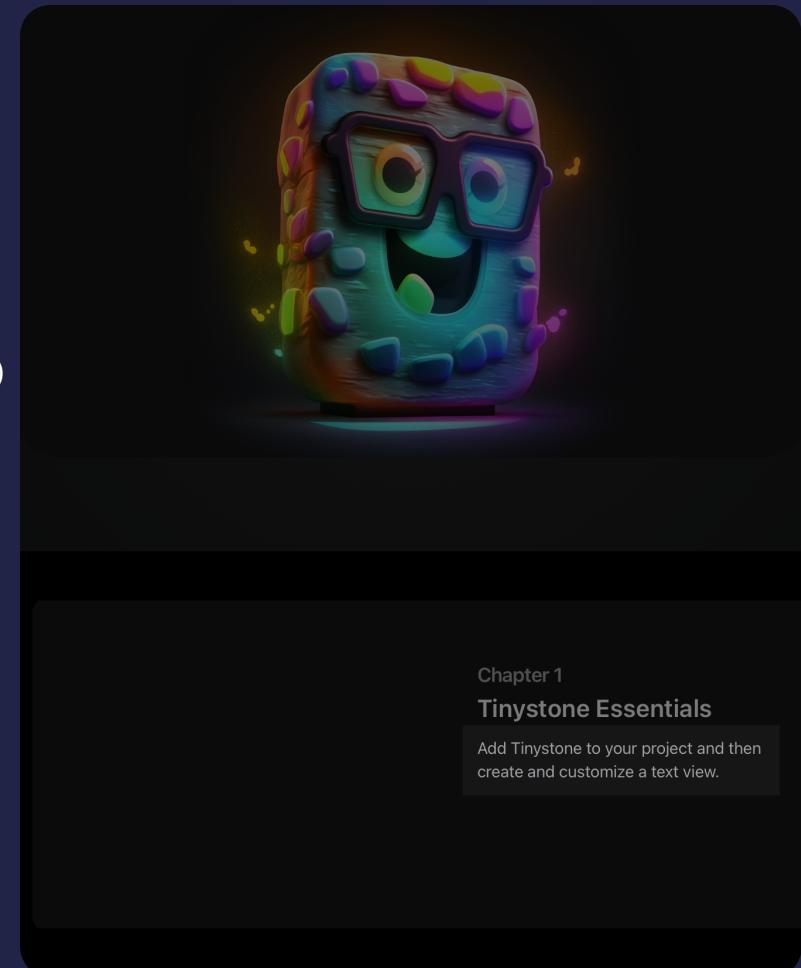
```
@Tutorials(name: "Tinystone") {
 @Intro(title: "Meet Tinystone") {
 Explore Tinystone, the performant code editor,
 through a series of interactive tutorials.

 @Image(source: hero.png, alt: "Happy stone in many colors")
 }

 @Chapter(name: "Tinystone Essentials") {
 Add Tinystone to your project and then create and
 customize a text view.

 @Image(source: file, alt: "accessible description")

 @TutorialReference(tutorial: "doc:tutorial name")
 }
}
```





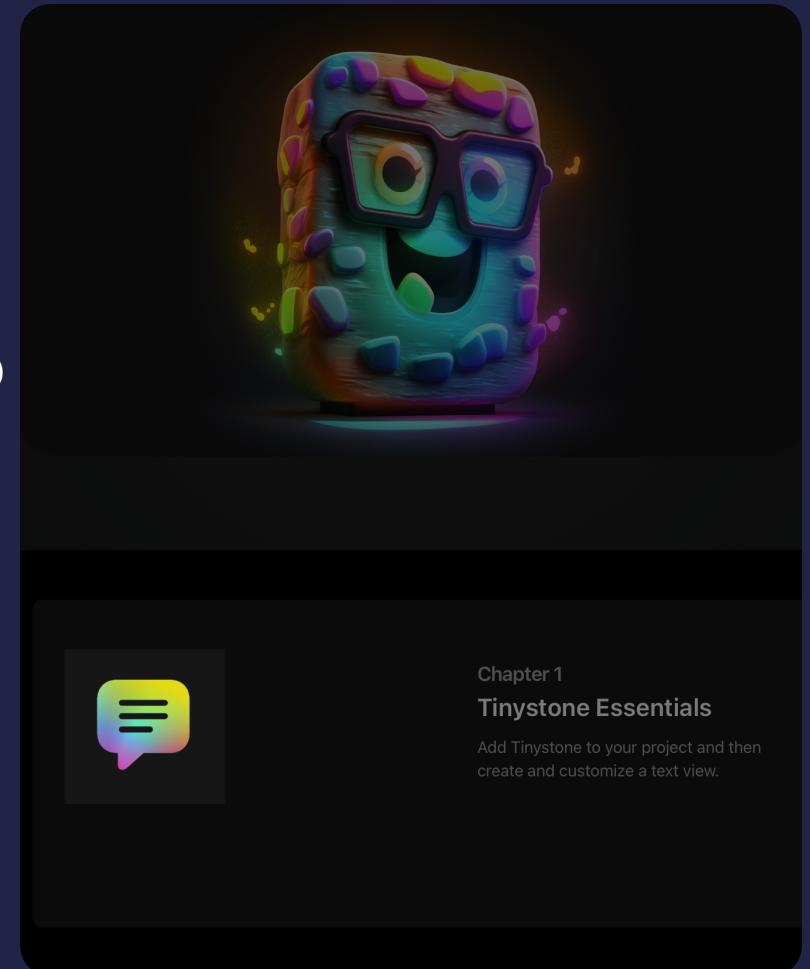
```
@Tutorials(name: "Tinystone") {
 @Intro(title: "Meet Tinystone") {
 Explore Tinystone, the performant code editor,
 through a series of interactive tutorials.

 @Image(source: hero.png, alt: "Happy stone in many colors")
 }

 @Chapter(name: "Tinystone Essentials") {
 Add Tinystone to your project and then create and
 customize a text view.

 @Image(source: chapter.png, alt: "Text forming an R")

 @TutorialReference(tutorial: "doc:tutorial name")
 }
}
```



Chapter 1  
**Tinystone Essentials**  
Add Tinystone to your project and then  
create and customize a text view.

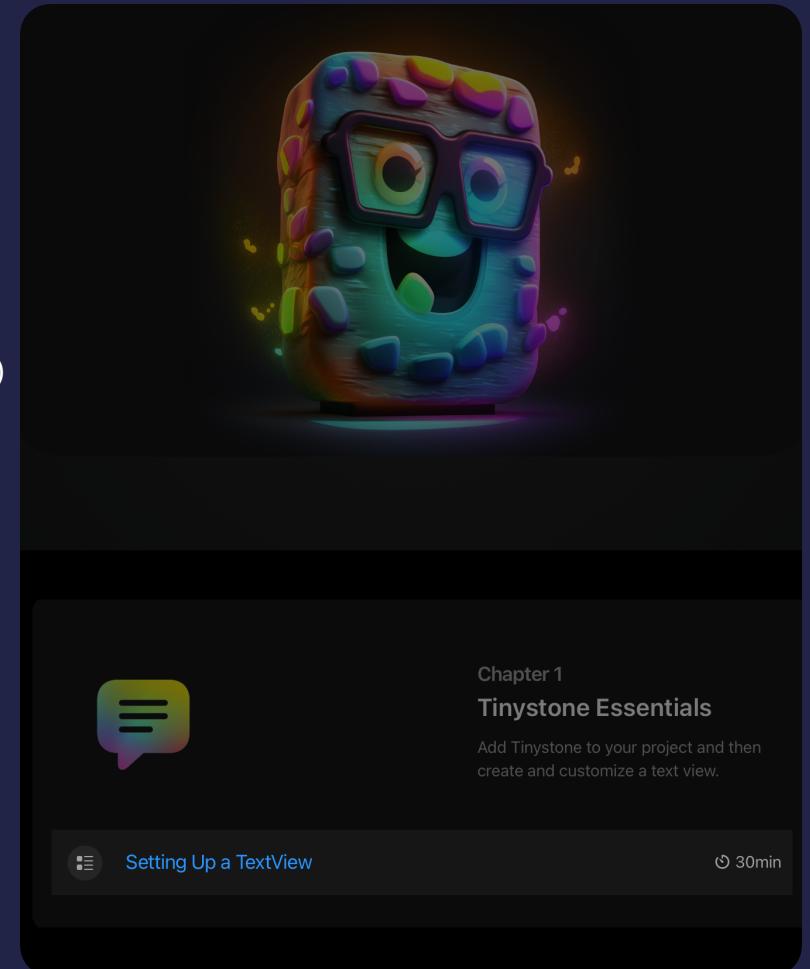


```
@Tutorials(name: "Tinystone") {
 @Intro(title: "Meet Tinystone") {
 Explore Tinystone, the performant code editor,
 through a series of interactive tutorials.

 @Image(source: hero.png, alt: "Happy stone in many colors")
 }

 @Chapter(name: "Tinystone Essentials") {
 Add Tinystone to your project and then create and
 customize a text view.

 @Image(source: chapter.png, alt: "Text forming an R")
 @TutorialReference(tutorial: "doc:Setting-Up-a-TextView")
 }
}
```



# TED LASSO





Describe Symbols



Add Structure



Add Supplemental  
Content



Describe Symbols



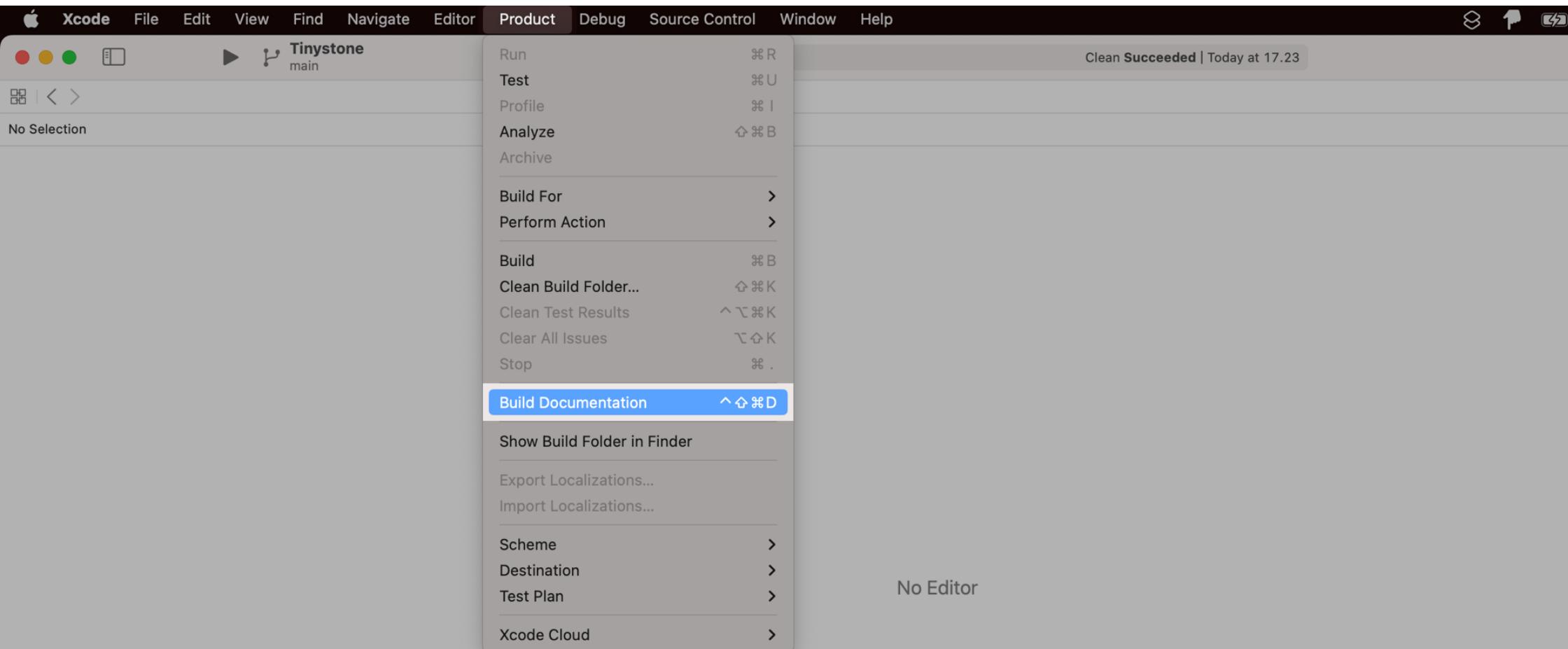
Add Structure



Add Supplemental  
Content



# How can people read my documentation?



The screenshot shows the Xcode documentation browser window. The sidebar on the left lists various Swift workspace documents, with "Tinystone" selected. The main content area displays the "Tinystone" framework documentation. The title "Tinystone" is prominently shown, followed by a description: "Performant code editor with syntax highlighting, line numbers, and invisible characters." A large, stylized blue graphic of overlapping rounded rectangles is visible on the right side of the main content area. Below the main content, there is a section titled "Overview" with a list of features.

**Framework**

# Tinystone

Performant code editor with syntax highlighting, line numbers, and invisible characters.

## Overview

Tinystone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlight the text.
- Show line numbers in the gutter.
- Showing invisible characters such as tabs, spaces, and line breaks.
- Adjust the height of lines.

A large, colorful illustration of a blocky, stylized character with a face, wearing large, round, colorful glasses. The character has a textured, colorful appearance with shades of blue, green, yellow, and pink. It is set against a dark background with some glowing particles around it.



```
xcodebuild docbuild \
-scheme Runestone \
-destination 'generic/platform=iOS' \
-derivedDataPath DerivedData
```

```
xcodebuild docbuild \
-scheme Runestone \
-destination 'generic/platform=iOS' \
-derivedDataPath DerivedData

$(xcrun --find docc) process-archive transform-for-static-hosting \
DerivedData/Build/Products/Debug-iphoneos/Runestone.doccarchive \
--output-path _site \
--hosting-base-path /
```



Swift Package Index.com

The screenshot shows the Swift Package Index website with the following details:

- Header:** The title "Swift Package Index" is displayed next to a red and orange logo. Navigation links include "Add a Package", "Blog", "FAQ", and "Search".
- Motto:** A message encourages users to "Do you maintain a package with DocC-compatible documentation? Let us build and host your docs!"
- Breadcrumbs:** The path "Home > Simon Størvring > Runestone" is shown.
- Section Headers:** "Runestone" and "simonbs/Runestone".
- Call-to-Action:** A blue button labeled "Use this Package".
- Description:** "Performant plain text editor for iOS with syntax highlighting, line numbers, invisible characters and much more."
- Development Status:**
  - A clock icon indicates it's "In development for almost 2 years, with 994 commits and 7 releases."
  - A speech bubble icon indicates there are 8 open issues and no open pull requests. The last issue was closed 3 hours ago and the last pull request was merged/closed 3 hours ago.
- Dependencies:** "This package has no package dependencies."
- Licensing:** "MIT licensed".
- Metrics:** "1,907 stars", "1 library", "No executables", and "No plugins".
- Tags:** "ios 887", "swift 2.4k", and "tree-sitter 3".
- Compatibility:** A chart showing compatibility across Swift versions 5.7, 5.6, 5.5, and 5.4. The 5.4 column is marked with a red "X". Below this, a chart shows compatibility across platforms: iOS (green checkmark), macOS (red X), watchOS (red X), tvOS (red X), and Linux (red X). Both charts are for version 0.2.3 and the main branch.
- Build Results:** "Full Build Results" section showing green checkmarks for all platforms.
- Branches:** "main" (Default Branch, modified 3 hours ago).
- Documentation:** A link to "Documentation".
- Notes:** A note asking if the user maintains the package, with instructions to get shields.io compatibility badges.
- Timestamp:** "Last updated on 23 Sep 2022".
- Footer:** Buttons for "README" and "Release Notes".
- Code Preview:** A snippet of Swift code from the README file, showing imports for TreeSitter and SwiftUI, and a class definition for `TextEditor`.

The screenshot shows a web browser displaying the Swift Package Index at [swiftpackageindex.com](https://swiftpackageindex.com). The page is for the package **Runestone**, maintained by [simonbs/Runestone](#). The package is described as a "Performant plain text editor for iOS with syntax highlighting, line numbers, invisible characters and much more." It has been in development for almost 2 years, with 994 commits and 7 releases. There are 8 open issues and no open pull requests. The package has no dependencies, is MIT licensed, contains 1 library, and has no executables, plugins, or shields.io badges. The latest stable release is 0.2.3, released 11 days ago. The main branch was modified 3 hours ago. The package is compatible with Swift 5.7, 5.6, 5.5, and 5.4, and runs on iOS, macOS, watchOS, tvOS, and Linux. The README and Release Notes tabs are visible, with the README tab currently active, showing a snippet of code related to tree-sitter and SwiftUI.

The screenshot shows a web browser displaying the Swift Package Index documentation for the `Runestone` framework version 0.2.3. The URL in the address bar is `swiftpackageindex.com`. The page title is `Swift Package Index > Simon Størring > Runestone > Documentation for 0.2.3`. The sidebar on the left contains a navigation tree for the `Runestone` framework, including sections like `Essentials`, `Appearance`, `Syntax Highlighting`, `Selected Line`, `Indentation`, `Search and Replace`, and `BatchReplaceSet`. The main content area features a large image of three nested, overlapping rectangles in shades of gray. The title `Runestone` is displayed in bold, followed by a description: "Performant code editor with syntax highlighting, line numbers, invisible characters, and much more." Below this, the `Overview` section lists various features with bullet points. A note states that the text view provided by Runestone does not subclass `UITextView` but has an API similar to it. The framework is available on [GitHub](#). At the bottom, there is a snippet of code from the `Textview` class.

```
1 let date = Date() // [Tree-sitter] (https://tree-sitter.io) - Line management is translated to Swift
2 let y = "" + date.getFullYear() // - Delegate to receive callback
3
4 public final class TextView: UIScrollView {
5 ...
6 }
```

The screenshot shows a web browser displaying the Swift Package Index documentation for the `Runestone` framework version `0.2.3`. The URL in the address bar is `swiftpackageindex.com`. The page title is `Documentation for 0.2.3`. On the left, there is a sidebar with a tree view of the API structure under the `Runestone` package. The main content area features a large image of three nested, overlapping rectangles in shades of gray. The title `Runestone` is displayed above a brief description: "Performant code editor with syntax highlighting, line numbers, invisible characters, and much more." Below the description, a section titled `Overview` lists various features of the framework. A note states: "The text view provided by Runestone does not subclass `UITextView` but has an API that is similar to the one of `UITextView`." At the bottom, it says "The framework is available on GitHub." A snippet of Swift code is shown at the very bottom.

Documentation / Runestone

Runestone

Framework

# Runestone

Performant code editor with syntax highlighting, line numbers, invisible characters, and much more.

## Overview

Runestone provides a text view with support for features commonly found in code editors. This includes the following list of features.

- Syntax highlighting.
- Line numbers.
- Highlight the selected line.
- Show invisible characters (tabs, spaces and line breaks).
- Insertion of character pairs, e.g. inserting the trailing quotation mark when inserting the leading.
- Customization of colors and fonts.
- Toggle line wrapping on and off.
- Adjust height of lines.
- Add a page guide.
- Add vertical and horizontal overscroll.
- Highlight ranges in the text view.
- Search the text using regular expressions.
- Automatically detects if a file is using spaces or tabs for indentation.
- Specify line endings (CR, LF, CRLF) to use when inserting a line break.
- Automatically detect line endings in a text.

The text view provided by Runestone does not subclass `UITextView` but has an API that is similar to the one of `UITextView`.

The framework is available on [GitHub](#).

```
[tree-sitter](https://tree-sitter.io) Line management is translated to Swift public final class TextView: UIScrollView { ... } // Delegate to receive callback
```

The screenshot shows a web browser displaying the Swift Package Index documentation for the `Runestone` package. The URL in the address bar is `swiftpackageindex.com`. The page title is `Documentation for 0.2.3`. A navigation bar at the top includes links for `main` and `0.2.3`. The main content area is titled `Framework` and features a large heading `Runestone`. Below the heading, a description reads: `Performant code editor with syntax highlighting, line numbers, invisible characters, and much more.`. A section titled `Overview` lists the following features:

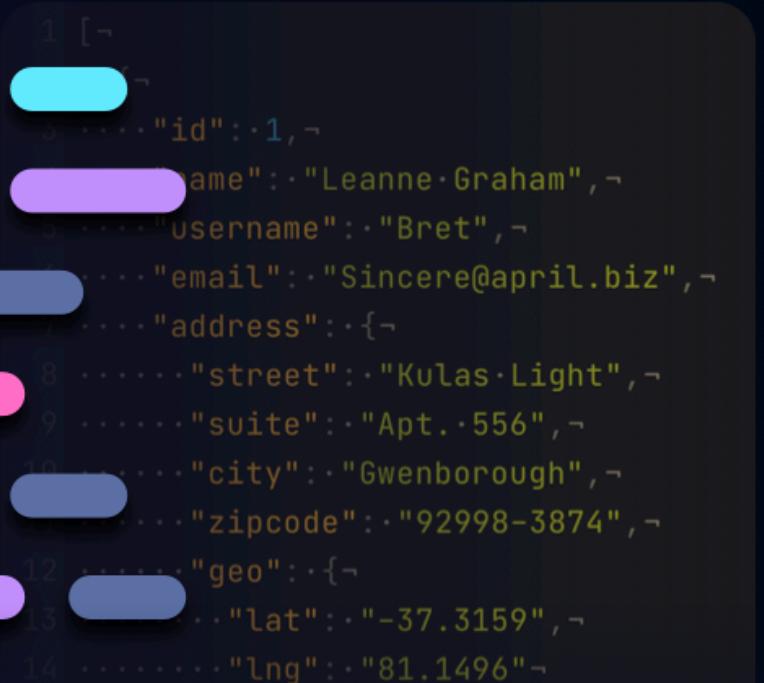
- Syntax highlighting.
- Line numbers.
- Highlight the selected line.
- Show invisible characters (tabs, spaces and line breaks).
- Insertion of character pairs, e.g. inserting the trailing quotation mark when inserting the leading.
- Customization of colors and fonts.
- Toggle line wrapping on and off.





```
Let y = ""+date.getFullYear()
Let m = ""+(date.getMonth() + 1)
Let d = ""+date.getDate()
Let dateStr = y + "-" + zeroPrefix(m) + "
Let siriArgs = args.siriShortcutArg
Let channelId = siriArgs.channel
Let channelIds = []
if (channelId != null) {
 channelIds = [channelId]
} else {
 channelIds = ["1", // DR1
 "3", // TV2
 "2", // DR2
]
 warning: 60
 error: 60
identifier_name:
allowed_symbols: "_"
max_length:
warning: 60
error: 60
```

```
38 - Line management is translated to Swift
39 - [swift-tree-sitter](https://github.com/tree-sitter/swift-tree-sitter)
40 - Detection of indent strategy inspired by Python's
41 - And last (but not least!), thanks to the new
```



```
1 [~
 2 ~
 3 ~
 4 ~
 5 "id": 1,
 6 "name": "Leanne Graham",
 7 "username": "Bret",
 8 "email": "Sincere@april.biz",
 9 "address": {
 10 "street": "Kulas Light",
 11 "suite": "Apt. 556",
 12 "city": "Gwenborough",
 13 "zipcode": "92998-3874",
 14 "geo": {
 15 "lat": "-37.3159",
 16 "lng": "81.1496"
 17 }
 18 },
 19 "phone": "1-770-736-8031 x56442"
 20 }
 21 }
 22 }
 23 }
 24 }
 25 public var text: String {
 26 get {
 27 return textView.string
 28 }
 29 set {
 30 textView.string = newValue
 31 contentSize = textView.contentSize
 32 }
 33 }
```

```
15 ... // Delegate to receive callback
16 public weak var editorDelegate:
17 ... // Whether the text view is in edit mode
18 public private(set) var isEditing:
19 didSet {
20 if isEditing != oldValue {
21 textView.isEditing = isEditing
22 }
23 }
24 }
25 public var text: String {
26 get {
27 return textView.string
28 }
29 set {
30 textView.string = newValue
31 contentSize = textView.contentSize
32 }
33 }
```

```
1 <!DOCTYPE html>
2 <html>
3 <meta charset="UTF-8" />
```



github.com/simonbs/runestone



**It has never been easier  
to make great documentation**



**SIMONBS.DEV**