Documenting Your Project With DocC

 @SIMONBS


@SIMONBS

How can I document my project?

@SIMONBS

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?

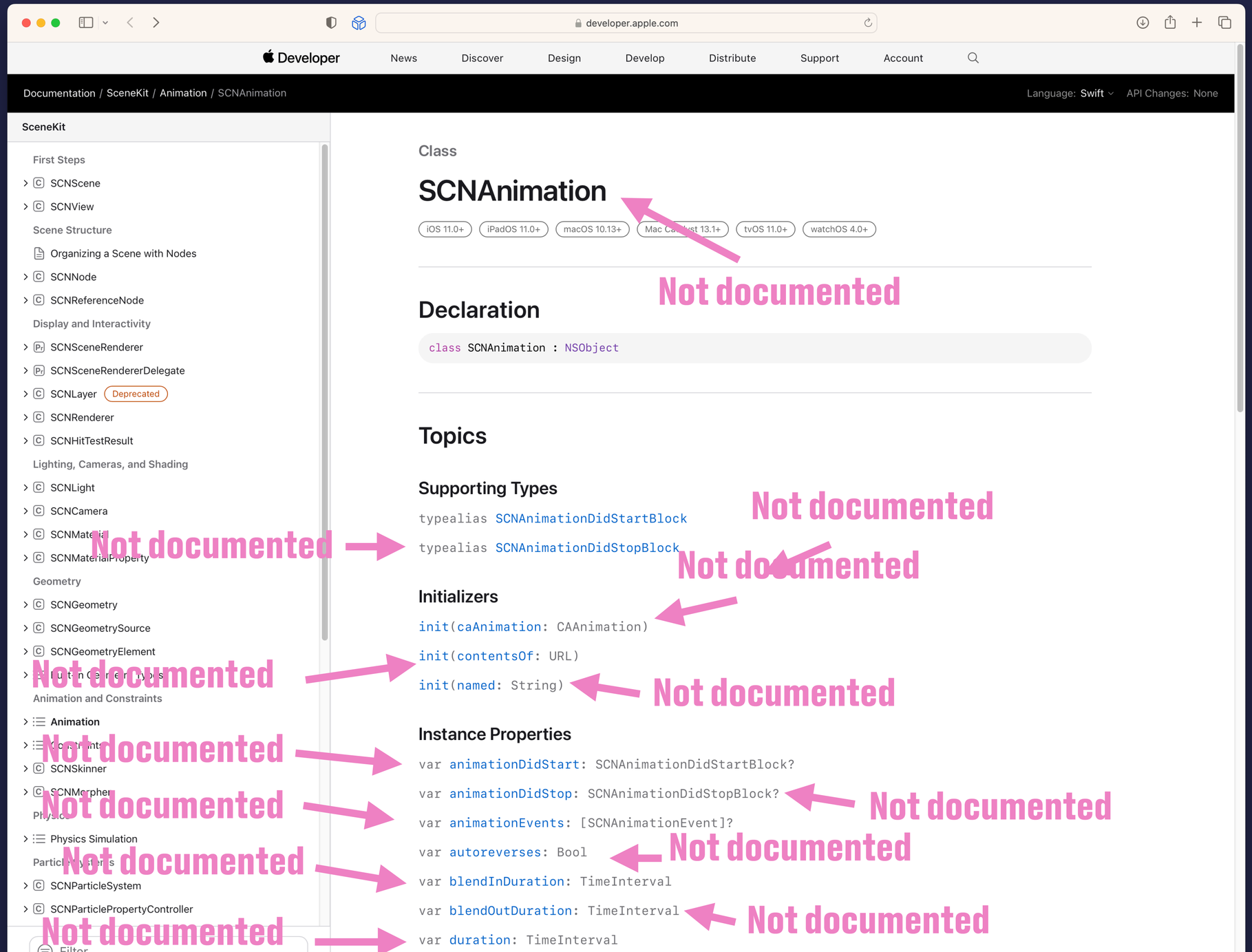
var animationEvents: [SCNAnimationEvent]?

var autoreverses: Bool

var blendInDuration: TimeInterval

var blendOutDuration: TimeInterval

var duration: TimeInterval


developer.apple.com
Developer
News
Discover
Design
Develop
Distribute
Support
Account
Documentation / SceneKit / Animation / SCNAnimation
Language: Swift
API Changes: None
SceneKit
First Steps
SCNScene
SCNView
Scene Structure
Organizing a Scene with Nodes
SCNNode
SCNReferenceNode
Display and Interactivity
SCNSceneRenderer
SCNSceneRendererDelegate
SCNLayer
Deprecated
SCNRenderer
SCNHitTestResult
Lighting, Cameras, and Shading
SCNLight
SCNCamera
SCNMaterial
SCNMaterialProperty
Geometry
SCNGeometry
SCNGeometrySource
SCNGeometryElement
Animation and Constraints
Animation
SCNSkinner
SCNMorpher
Physics Simulation
SCNParticleSystem
SCNParticlePropertyController
Class
SCNAnimation
iOS 11.0+
iPadOS 11.0+
macOS 10.13+
tvOS 11.0+
watchOS 4.0+
Not documented
Declaration
class SCNAnimation : NSObject
Topics
Supporting Types
typealias SCNAnimationDidStartBlock
typealias SCNAnimationDidStopBlock
Not documented
Not documented
Not documented
Initializers
init(caAnimation: CAAnimation)
init(contentsOf: URL)
init(named: String)
Not documented
Not documented
Not documented
Not documented
Instance Properties
var animationDidStart: SCNAnimationDidStartBlock?
var animationDidStop: SCNAnimationDidStopBlock?
var animationEvents: [SCNAnimationEvent]?
var autoreverses: Bool
var blendInDuration: TimeInterval
var blendOutDuration: TimeInterval
var duration: TimeInterval
Not documented
Not documented
Not documented
Not documented
Not documented


@SIMONBS

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

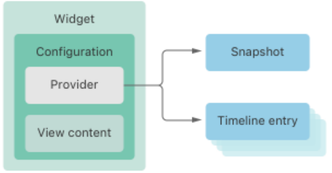
WidgetKit gives users ready access to content in your app by putting widgets on the iOS Home Screen and Today View, the macOS Notification Center, and by putting accessory widgets on the Lock Screen on iPhone and as complications in watchOS. Your widgets stay up to date so users always have the latest 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.



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

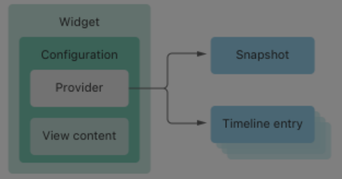
WidgetKit gives users ready access to content in your app by putting widgets on the iOS Home Screen and Today View, the macOS Notification Center, and by putting accessory widgets on the Lock Screen on iPhone and as complications in watchOS. Your widgets stay up to date so users always have the latest 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.



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

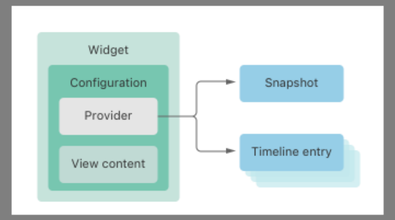
WidgetKit gives users ready access to content in your app by putting widgets on the iOS Home Screen and Today View, the macOS Notification Center, and by putting accessory widgets on the Lock Screen on iPhone and as complications in watchOS. Your widgets stay up to date so users always have the latest 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.



Add and configure an extension to show your app's content on the Home Screen, Today View, or Notification Center.

Creating Lock Screen Widgets and Watch Complications

Create accessory widgets that appear on the iPhone Lock Screen and as complications on Apple Watch.

Migrating ClockKit complications to WidgetKit

Leverage WidgetKit's API to create watchOS complications using SwiftUI.

{} Building Widgets Using WidgetKit and SwiftUI

Create widgets to show your app's content on the Home screen, with custom intents for user-customizable settings.

{} Adding widgets to the Lock Screen and watch faces

Discover how to create iPhone Lock Screen widgets and watchOS complications with WidgetKit.

{} Fruta: Building a Feature-Rich App with SwiftUI

Create a shared codebase to build a multiplatform app that offers widgets and an App Clip.

`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.

Configurable widgets

Making a Configurable Widget

Give users the option to customize their widgets by adding a custom SiriKit intent definition to your project.

`struct` **`IntentConfiguration`**

An object describing the content of a widget that uses a custom intent definition to provide user-configurable options.

`struct` **`WidgetInfo`**

A structure that contains information about user-configured widgets.

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.

 @SIMONBS

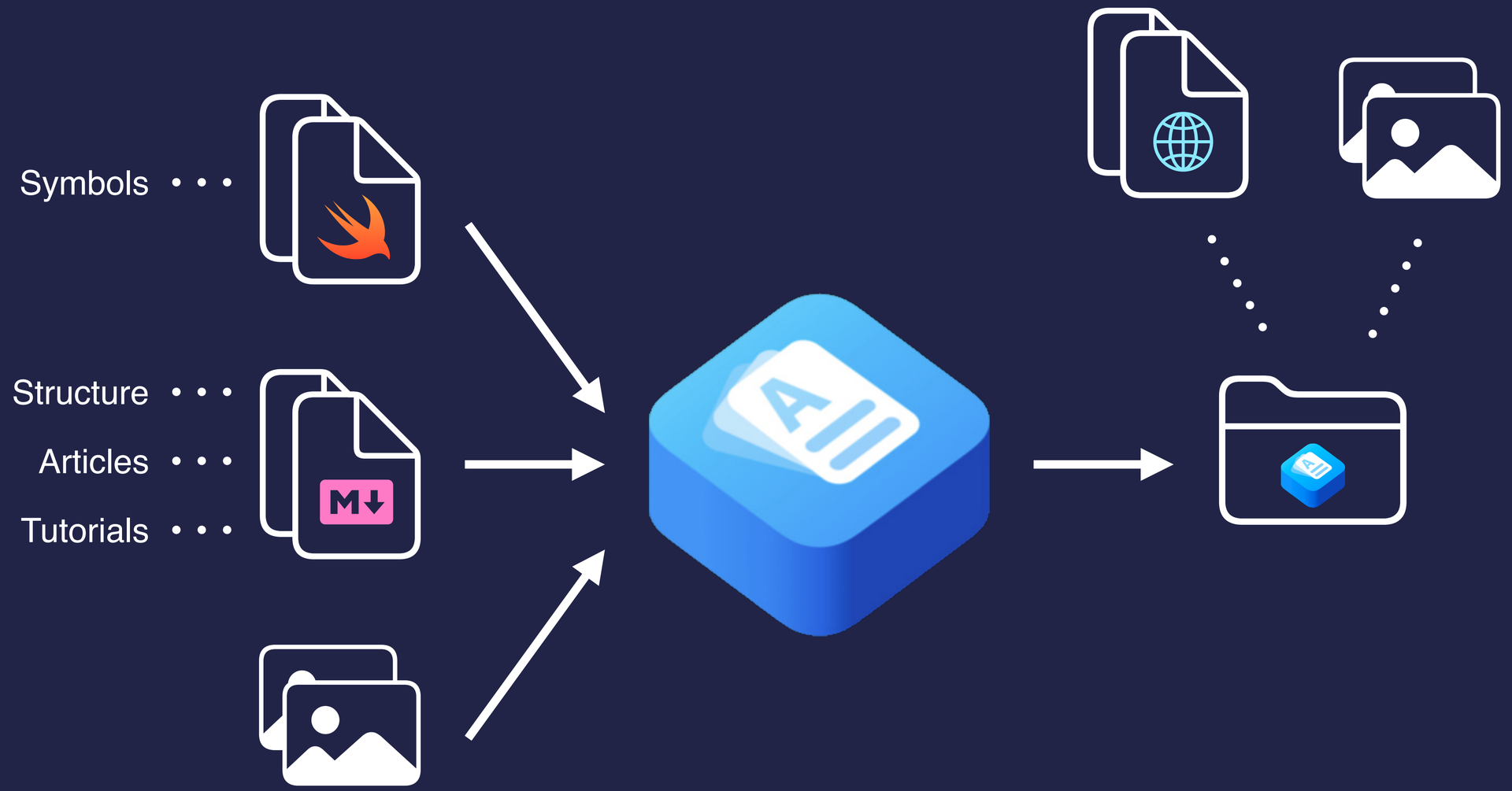
Apple’s documentation
is great when it exists.

 @SIMONBS

@SIMONBS

@SIMONBS


Chapter 1 Title
Symbol
Tutorial Title
@SIMONBS


Symbols
Structure
Articles
Tutorials

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


@SIMONBS

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

@SIMONBS

Instance Method

search(for:)

Declaration

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

@SIMONBS

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

@SIMONBS

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

@SIMONBS

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

 @SIMONBS

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

@SIMONBS

```
/// 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", matchMethod: .contains)
/// let results = textView.search(for: query)
/// ```
///
/// - Parameter query: Query to find matches for.
/// - Returns: Results matching the query.
func search(for query: SearchQuery) -> [SearchResult]
```

@SIMONBS

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

 @SIMONBS

Instance Method

search(for:)

Search for the specified query.

Declaration

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

Return Value

Results matching the query.

Parameters

query
Query to find matches for.

Discussion

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

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

> Note
> A string of size N can be search in time O(N).

@SIMONBS

Framework

Runestone

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

Topics

Articles

- Adding a Tree-sitter Language

  Learn how to add one of Tree-sitter's many supported language and start syntax highlighting text.

- Creating a Theme

  Learn how to create a theme and customize the appearance of TextView.

- Getting Started

  This articles gets you started using Runestone in your project.

Classes

`class` `DefaultTheme`

Default theme used by Runestone when no other theme has been set.

`class` `HighlightedRange`

Range of text to highlight.

`class` `PlainTextLanguageMode`

Language mode with no syntax highlighting.

`class` `TextPreview`

Provides a peek into the underlying attributed string of a text view.

`class` `TextView`

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

`class` `TextViewState`

Encapsulates the bare informations needed to do syntax highlighting in a text view.

`class` `TreeSitterIndentationScopes`

Indentation rules to be used with a `TreeSitterLanguage`.

`class` `TreeSitterLanguage`

Language to use for syntax highlighting with Tree-sitter.

`class` `TreeSitterLanguageMode`

Perform syntax highlighting with Tree-sitter.

@SIMONBS

Framework

Runestone

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

Topics

Articles

- Adding a Tree-sitter Language

  Learn how to add one of Tree-sitter's many supported language and start syntax highlighting text.

- Creating a Theme

  Learn how to create a theme and customize the appearance of TextView.

- Getting Started

  This articles gets you started using Runestone in your project.

Classes

`class DefaultTheme`

Default theme used by Runestone when no other theme has been set.

`class HighlightedRange`

Range of text to highlight.

`class PlainTextLanguageMode`

Language mode with no syntax highlighting.

`class TextPreview`

Provides a peek into the underlying attributed string of a text view.

`class TextView`

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

`class TextViewState`

Encapsulates the bare informations needed to do syntax highlighting in a text view.

`class TreeSitterIndentationScopes`

Indentation rules to be used with a `TreeSitterLanguage`.

`class TreeSitterLanguage`

Language to use for syntax highlighting with Tree-sitter.

`class TreeSitterLanguageMode`

Perform syntax highlighting with Tree-sitter.

@SIMONBS

Topics

Articles

- Adding a Tree-sitter Language
  Learn how to add one of Tree-sitter's many supported language and start syntax highlighting text.
- Creating a Theme
  Learn how to create a theme and customize the appearance of TextView.
- Getting Started
  This articles gets you started using Runestone in your project.

Classes

`class DefaultTheme`
Default theme used by Runestone when no other theme has been set.

`class HighlightedRange`
Range of text to highlight.

`class PlainTextLanguageMode`
Language mode with no syntax highlighting.

`class TextPreview`
Provides a peek into the underlying attributed string of a text view.

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

`class TextViewState`
Encapsulates the bare informations needed to do syntax highlighting in a text view.

`class TreeSitterIndentationScopes`
Indentation rules to be used with a `TreeSitterLanguage`.

`class TreeSitterLanguage`
Language to use for syntax highlighting with Tree-sitter.

`class TreeSitterLanguageMode`
Perform syntax highlighting with Tree-sitter.

Protocols

`protocol CharacterPair`
Character pair to be registered with a text view.

`protocol LanguageMode`
Mode used for syntax highlighting text in the text view.

`protocol TextViewDelegate`


@SIMONBS

```
## Topics

### Essentials

### Appearance

### Syntax Highlighting
```

@SIMONBS

```
## Topics

### Essentials

- ``TextView``

- ``TextViewDelegate``

- ``TextViewState``

### Appearance

- ``Theme``

- ``FontTraits``

### Syntax Highlighting

- ``PlainTextLanguageMode``

- ``TreeSitterLanguageMode``
```

 @SIMONBS

Framework

Runestone

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

Topics

Essentials

Meet Runestone

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

Setting up a TextView

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

Getting Started

This articles gets you started using Runestone in your project.

`class TextView`

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

`protocol TextViewDelegate`

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

`class TextViewState`

Encapsulates the bare informations needed to do syntax highlighting in a text view.

Appearance

The appearance of `TextView` can be customized using a `Theme`. This is also for the text styling and colors used for syntax highlighting.

Adding a Theme

This tutorial guides you through adding a custom theme to your project. You will add the Tomorrow theme to the TextCompanion app and use it when highlighting code.

Creating a Theme

Learn how to create a theme and customize the appearance of TextView.

`protocol Theme`

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

`struct FontTraits`

Traits to be applied to a font.

`class DefaultTheme`

@SIMONBS

Framework

Runestone

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

Topics

Essentials

Meet Runestone

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

Setting up a TextView

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

Getting Started

This articles gets you started using Runestone in your project.

`class TextView`

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

`protocol TextViewDelegate`

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

`class TextViewState`

Encapsulates the bare informations needed to do syntax highlighting in a text view.

Appearance

The appearance of `TextView` can be customized using a `Theme`. This is also for the text styling and colors used for syntax highlighting.

Adding a Theme

This tutorial guides you through adding a custom theme to your project. You will add the Tomorrow theme to the TextCompanion app and use it when highlighting code.

Creating a Theme

Learn how to create a theme and customize the appearance of TextView.

`protocol Theme`

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

`struct FontTraits`

Traits to be applied to a font.

`class DefaultTheme`

@SIMONBS

Essentials

Meet Runestone

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

Setting up a TextView

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

Getting Started

This articles gets you started using Runestone in your project.

`class TextView`

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

`protocol TextViewDelegate`

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

`class TextViewState`

Encapsulates the bare informations needed to do syntax highlighting in a text view.

Appearance

The appearance of `TextView` can be customized using a `Theme`. This is also for the text styling and colors used for syntax highlighting.

Adding a Theme

This tutorial guides you through adding a custom theme to your project. You will add the Tomorrow theme to the TextCompanion app and use it when highlighting code.

Creating a Theme

Learn how to create a theme and customize the appearance of TextView.

`protocol Theme`

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

`struct FontTraits`

Traits to be applied to a font.

`class DefaultTheme`

Default theme used by Runestone when no other theme has been set.

`enum LineBreakMode`

Line break mode for text view.

`enum LineSelectionDisplayType`

Approach for highlighting the selected line.

Syntax Highlighting

Syntax highlighting is based on GitHub's parser generator. It has support for most popular programming languages.

Syntax Highlighting the Text

This tutorial guides you through syntax highlighting text in a text view.

@SIMONBS

Framework

Runestone

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

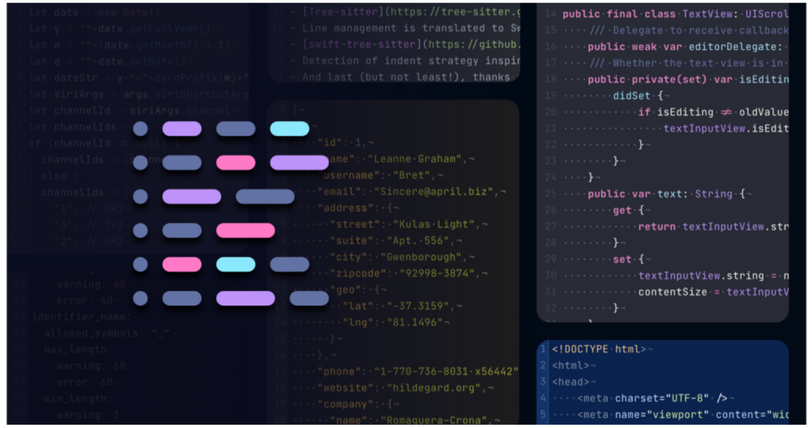
Overview

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.

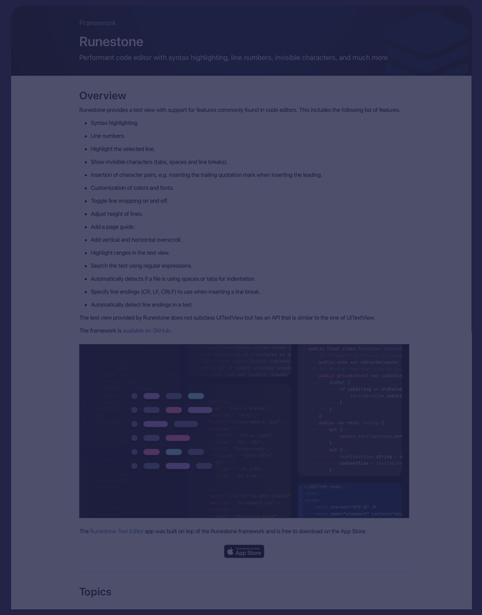
The framework is available on GitHub.



The Runestone Text Editor app was built on top of the Runestone framework and is free to download on the App Store.

Download on the App Store

@SIMONBS



Landing Page

Class

TextView

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

Declaration

```
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.

The type does not subclass `UITextView` but its interface is kept close to `UITextView`.

When initially configuring the `TextView` with a theme, a language and the text to be shown, it is recommended to use the `setState(_:addUndoAction:)` function. The function takes an instance of `TextViewState` as input which can be created on a background queue to avoid blocking the main queue while doing the initial parse of a text.

Topics

Initialing the Text View

`init(frame: CGRect)`

Create a new text view.

`init?(coder: NSCoder)`

The initializer has not been implemented.

Responding to Text View Changes

`var editorDelegate: TextViewDelegate?`

Delegate to receive callbacks for events triggered by the editor.

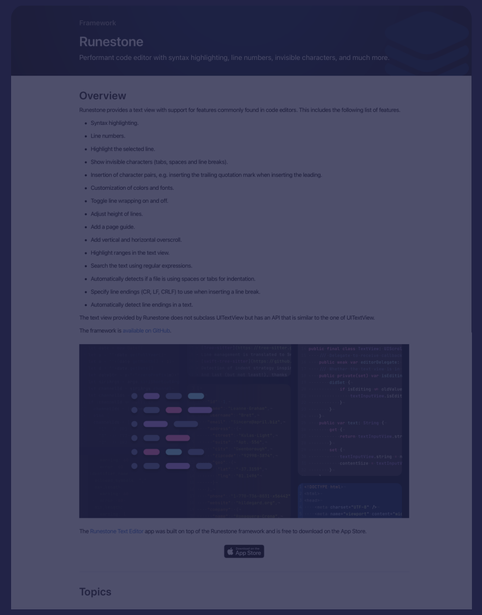
`protocol TextViewDelegate`

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

Configuring the Appearance

`var theme: Theme`

@SIMONBS



Landing Page

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: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

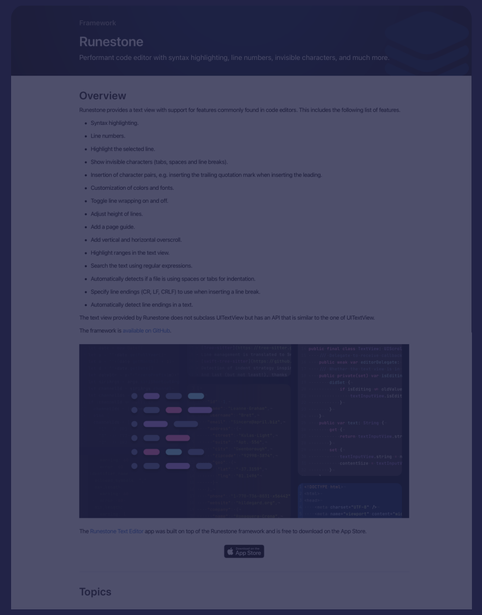
The `TextView` is a subclass of UIScrollView and as such can be initialized like any other view. It has an API that is very similar to the one of UITextView.

```
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%
```

@SIMONBS



Landing Page

Section 2

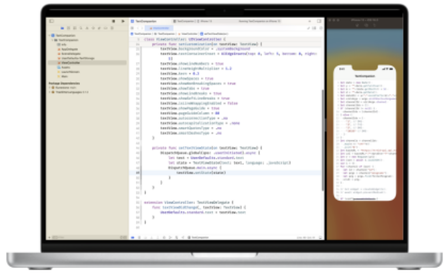
Setting the state with TextViewState

Up until now we have set the text and language on the text view using the `text` property and and `setLanguageMode(_:)` functions. Setting the text or language on the text view requires an analysis to be performed on the text. This analysis can be an expensive operation for large documents.

Because the `text` property and `setLanguageMode(_:)` function must be called on the main queue, the UI may hang for a while when calling any of those.

We can use `TextViewState`, a type that encapsulates the text, theme, and language, to avoid blocking the main queue. `TextViewState` will perform the analysis of the document when initialized and the state can be initialized on a background queue.

The following section explains how to use `TextViewState` to improve the performance of the TextCompanion app.



Step 1

Select the ViewController.swift file in the Project navigator.

Step 2

Add a function called `setTextViewState(on:)`.

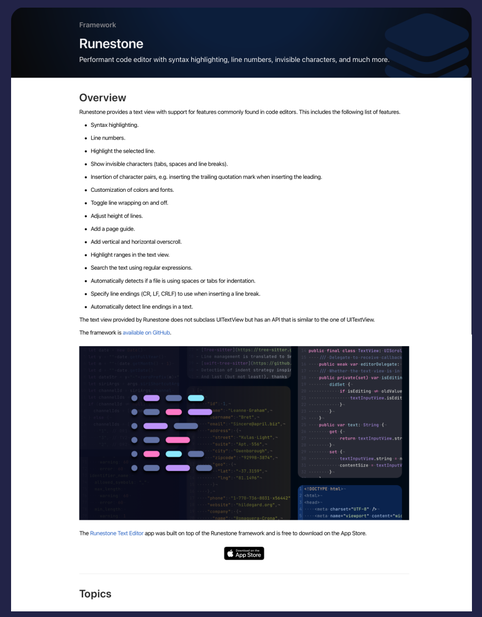
We will add the implementation of this function later.

ViewController.swift

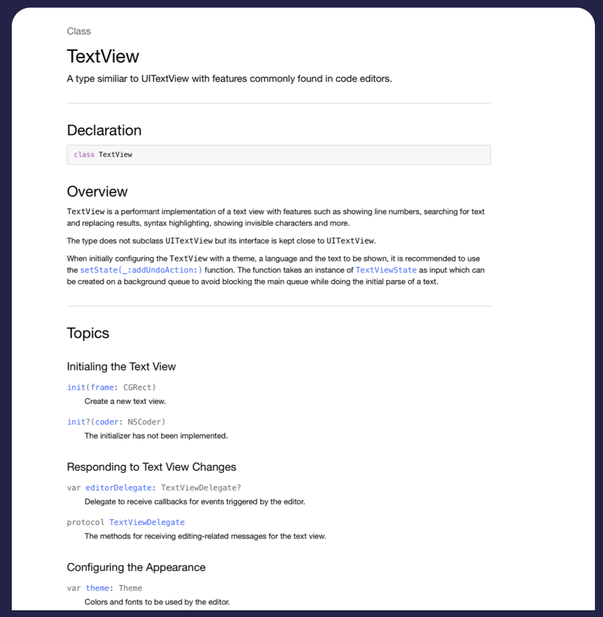
```swift
import Runestone
import TreeSitterJavaScriptRunestone
import UIKit

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        title = "TextCompanion"
        navigationController?.navigationBar.scrollEdgeAppearance = UINavigationBarAppearance()
        let textView = TextView()
        textView.translatesAutoresizingMaskIntoConstraints = false
        textView.editorDelegate = self
        textView.backgroundColor = .systemBackground
        textView.text = UserDefaults.standard.text
        let languageMode = TreeSitterLanguageMode(language: .javaScript)
        textView.setLanguageMode(languageMode)
        setCustomization(on: textView)
        view.addSubview(textView)
        NSLayoutConstraint.activate([
            textView.leadingAnchor.constraint(equalTo: view.leadingAnchor),
            textView.trailingAnchor.constraint(equalTo: view.trailingAnchor),
            textView.topAnchor.constraint(equalTo: view.topAnchor),
```

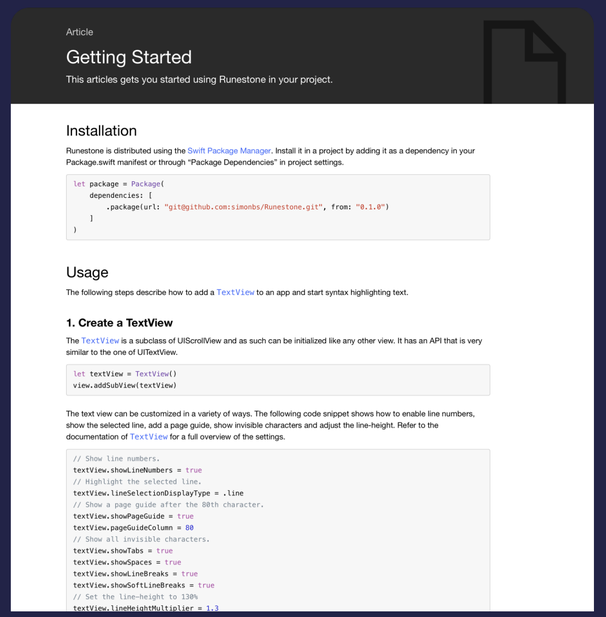
@SIMONBS



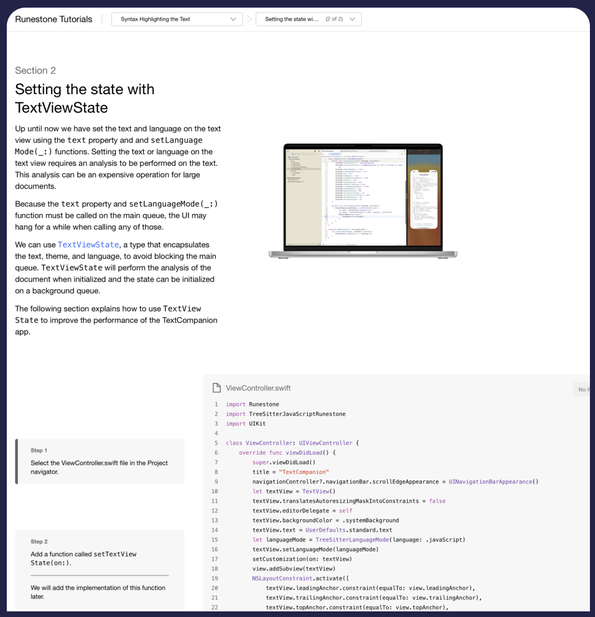
Landing Page



Extensions



Articles

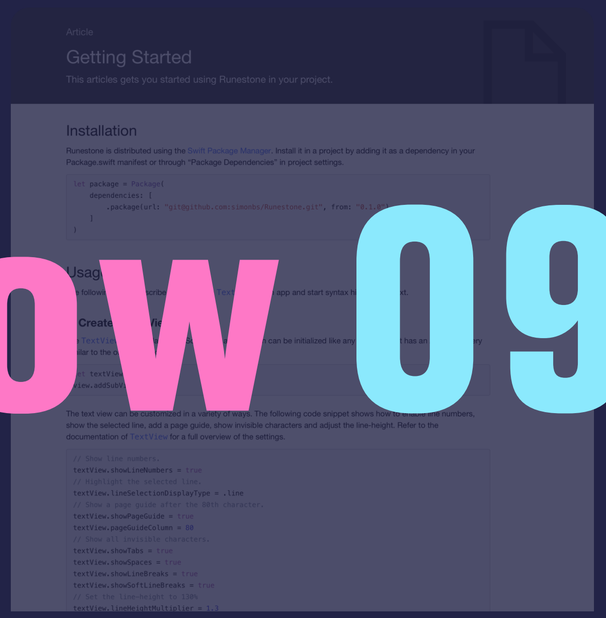


Tutorials

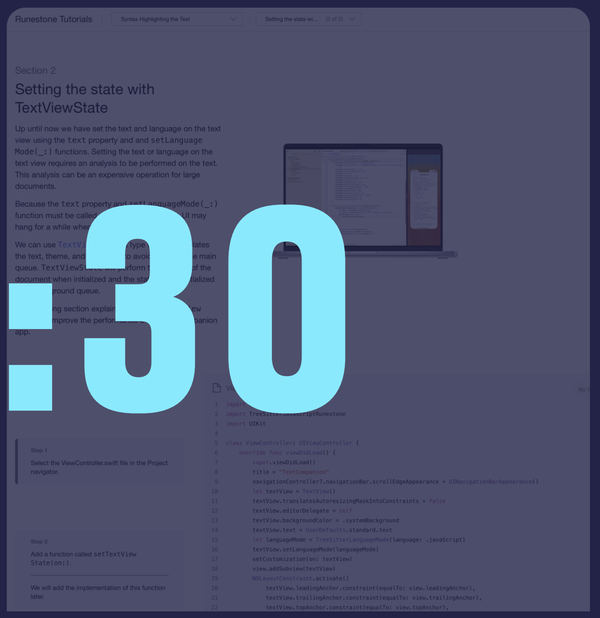
@SIMONBS

Landing Page

Extensions



Articles



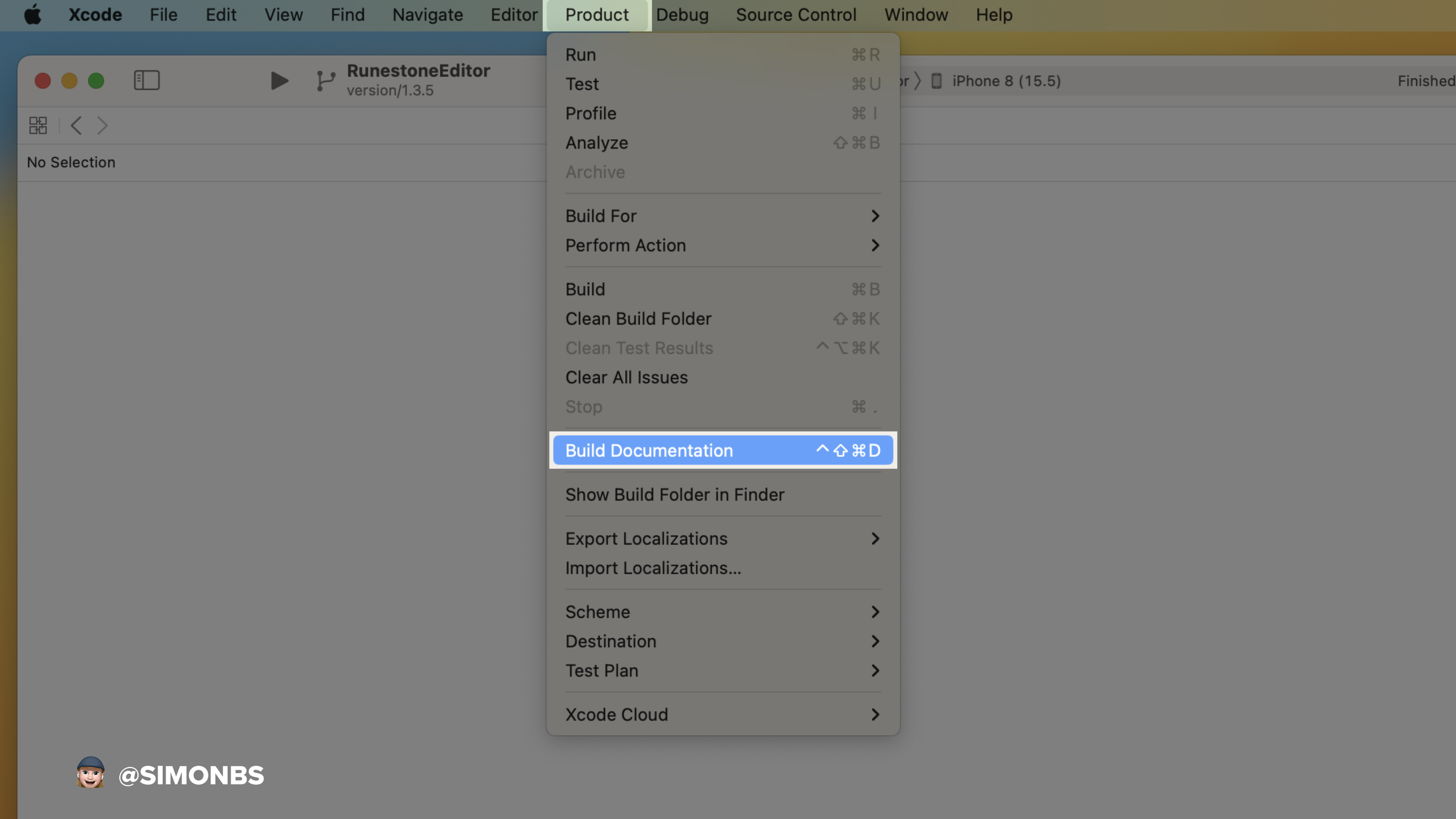
Tutorials

Tomorrow 09:30

@SIMONBS

How can people read my documentation?

 @SIMONBS


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
RunestoneEditor
version/1.3.5
iPhone 8 (15.5)
Finished
No Selection
Run ⌘R
Test ⌘U
Profile ⌘I
Analyze ⇧⌘B
Archive
Build For
Perform Action
Build ⌘B
Clean Build Folder ⇧⌘K
Clean Test Results ^⌥⌘K
Clear All Issues
Stop ⌘.
Build Documentation ^⇧⌘D
Show Build Folder in Finder
Export Localizations
Import Localizations...
Scheme
Destination
Test Plan
Xcode Cloud
@SIMONBS

Framework

Runestone

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

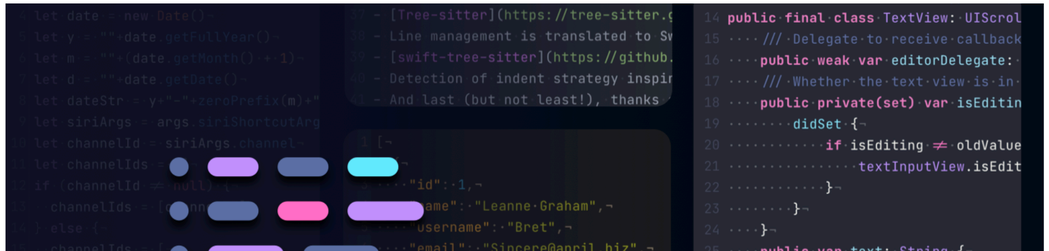
Overview

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.



GitHub

+

Netlify

=

docs.runestone.app

@SIMONBS

It has never been easier to make great documentation

 @SIMONBS

@SIMONBS


BuildABuddyKit

Join the classroom

Tomorrow

09:30

@SIMONBS