Swift Documentation Guide

A comprehensive guide to writing rich, professional documentation for your Swift code using Xcode's Quick Help system.

Table of Contents

Introduction

When you **Option + Click** any symbol in Xcode, you see the **Quick Help** window. This shows documentation that's either:

- Generated automatically from your code
- Written by you using special comment syntax

Adding documentation comments makes your code:

- ✅ Easier to understand for other developers (and future you)
- ✅ More professional and maintainable
- ✅ Accessible via Xcode's Quick Help system
- ✅ Ready for framework distribution

Basic Syntax

There are two ways to write documentation comments:

Single-Line Documentation Comments

Use three forward slashes (`///`):

```swift
/// This is a documentation comment
/// You can use multiple lines
func myFunction() {
    // Regular comment (not documentation)
}
```

Multi-Line Documentation Comments

Use `/** ... */`:

```swift
/**
 This is a multi-line documentation comment

 You can write longer descriptions here
 across multiple lines.
 */
func myFunction() {
}
```

Best Practice: Use `///` for most documentation. It's cleaner and easier to maintain.

Documentation Keywords

Swift documentation supports special keywords that format information nicely in Quick Help:

Core Keywords

Keyword	Purpose	Usage
`- Parameters:`	Document function parameters	Required for functions with parameters
`- Returns:`	Document return value	Required for functions that return
`- Throws:`	Document thrown errors	Required for throwing functions
`- Parameter <name>:`	Document single parameter	Alternative to Parameters block

Callout Keywords

Keyword	Icon	Purpose
`- Note:`	ℹ️	Additional information
`- Important:`	⚠️	Critical information
`- Warning:`	⚠️	Warning about potential issues
`- Tip:`	💡	Helpful suggestion
`- Experiment:`	🧪	Suggestion to try something
`- SeeAlso:`	🔗	Related symbols or resources
`- Precondition:`	✓	Requirements before calling
`- Postcondition:`	✓	State after calling
`- Requires:`	✓	System requirements
`- Invariant:`	=	Conditions that must hold
`- Complexity:`	⏱	Time/space complexity
`- TODO:`	☑️	Work to be done
`- Bug:`	🐛	Known issues
`- Version:`	📋	Version information
`- Author:`	👤	Author information
`- Copyright:`	©	Copyright notice
`- Date:`	📅	Creation/modification date
`- Since:`	📅	Version when introduced

Documenting Functions and Methods

Basic Function Documentation

swift

```swift
/// Calculates the sum of two numbers
///
/// - Parameters:
///   - a: The first number to add
///   - b: The second number to add
/// - Returns: The sum of `a` and `b`
func add(_ a: Int, _ b: Int) -> Int {
    return a + b
}
```

Function with Throws

swift

```swift
/// Divides two numbers
///
/// - Parameters:
///   - dividend: The number to be divided
///   - divisor: The number to divide by
/// - Returns: The result of the division
/// - Throws: `DivisionError.divideByZero` if divisor is zero
func divide(_ dividend: Double, by divisor: Double) throws -> Double {
    guard divisor != 0 else {
        throw DivisionError.divideByZero
    }
    return dividend / divisor
}
```

Comprehensive Function Documentation

swift

```swift
/// Presents a toast notification overlay
///
/// Shows a temporary message banner that slides in from the top of the screen,
/// automatically dismisses after the specified duration, and includes haptic feedback.
///
/// - Parameters:
///   - isPresented: When `true`, the toast appears. Automatically resets to `false` after dismissal.
///   - message: The text to display in the toast notification
///   - icon: SF Symbol name for the leading icon (default: `"checkmark.circle.fill"`)
///   - color: Accent color for the icon and border (default: `.green`)
///   - duration: Time in seconds before auto-dismiss (default: `2.0`)
///
/// - Returns: A view with the toast overlay modifier applied
///
/// # Example
/// ```swift
/// .toast(
///     isPresented: $showSuccess,
///     message: "Task completed!",
///     icon: "checkmark.circle.fill",
///     color: .green
/// )
/// ```
///
/// - Note: The toast includes haptic feedback on iOS devices
/// - Important: Only one toast should be active at a time
/// - Tip: Keep messages under 50 characters for best readability
func toast(
    isPresented: Binding<Bool>,
    message: String,
    icon: String = "checkmark.circle.fill",
    color: Color = .green,
    duration: TimeInterval = 2.0
```

```
) -> some View {
    modifier(ToastModifier(
        isPresented: isPresented,
        message: message,
        icon: icon,
        color: color,
        duration: duration
    ))
}
```

Alternative Single-Parameter Syntax

For simple functions, you can use `- Parameter`:

```swift
/// Prints a greeting message
///
/// - Parameter name: The name of the person to greet
func greet(name: String) {
    print("Hello, \(name)!")
}
```

Documenting Classes and Structs

Basic Class Documentation

```swift
```

```
/// A manager that handles CloudKit sharing operations
///
/// This class coordinates the creation and management of CKShares for tasks,
/// including updating TaskItem properties when shares are created or accepted.
///
/// - Important: Always inject a ModelContext before using
class CloudKitSharingManager: ObservableObject {
    // ...
}
```

Comprehensive Class Documentation with Topics

```
swift
```

```
/// A manager that handles CloudKit sharing operations
///
/// This class coordinates the creation and management of CKShares for tasks,
/// including updating TaskItem properties when shares are created or accepted.
///
/// ## Overview
/// The sharing manager provides a high-level interface for CloudKit sharing,
/// abstracting away the complexity of CKShare creation, participant management,
/// and share acceptance.
///
/// ## Topics
/// ### Creating Shares
/// - ``shareTask(_:taskName:completion:)``
/// - ``acceptShare(metadata:)``
///
/// ### Managing Shares
/// - ``stopSharing(taskId:completion:)``
/// - ``cancelShare(taskId:share:completion:)``
///
/// ### Properties
/// - ``activeShare``
/// - ``activeRecord``
/// - ``isLoading``
/// - ``sharingError``
///
/// ## Usage
/// ```swift
/// let manager = CloudKitSharingManager.shared
/// manager.modelContext = modelContext
///
/// manager.shareTask(taskId, taskName: "My Task") { result in
///     switch result {
///     case .success(let share):
```

```
///         print("Share created: \(share.url)")
///     case .failure(let error):
///         print("Error: \(error)")
///     }
/// }
/// ```
///
/// - Important: Always inject a ModelContext before using
/// - Note: This class is designed for @MainActor operation
/// - SeeAlso: ``CloudKitHelper``
@MainActor
class CloudKitSharingManager: ObservableObject {
    // ...
}
```

Struct Documentation

```
swift
```

```
/// A notification banner that appears temporarily at the top of the screen
///
/// `ToastView` displays a message with an icon in a frosted glass container.
/// It's typically used for brief confirmations like "Email sent" or "Task completed".
///
/// The view automatically includes:
/// - Material background with blur effect
/// - Colored icon and border
/// - Smooth spring animation
/// - Shadow for depth
///
/// - Important: This view should be presented via the ``toast(isPresented:message:icon:color:duration:)`` modifier,
///   not instantiated directly.
///
/// ## Example Appearance
/// ```
/// ┌─────────────────────────────────┐
/// │ ✓ Thank you for feedback! │
/// └─────────────────────────────────┘
/// ```
struct ToastView: View {
    /// The message text to display
    let message: String

    /// SF Symbol name for the icon
    let icon: String

    /// The accent color for the icon and border
    let color: Color

    var body: some View {
        // ...
```

```
    }
}
```

Documenting Properties

Basic Property Documentation

```swift
/// The currently active share being displayed
@Published var activeShare: CKShare?
```

Comprehensive Property Documentation

```swift
/// The currently active share being displayed
///
/// This property is set when a share is successfully created or fetched,
/// and is used by the UI to present share options to the user.
///
/// The property lifecycle:
/// 1. Set to a CKShare when sharing begins
/// 2. Passed to UICloudSharingController for presentation
/// 3. Cleared when sharing is cancelled or completed
///
/// - Note: This property is automatically cleared when sharing is cancelled
/// - Important: Should only be accessed on the main actor
@Published var activeShare: CKShare?
```

Computed Property Documentation

swift

```swift
/// The current app version including build number
///
/// Returns a string in the format "1.0 (5)" where 1.0 is the version
/// and 5 is the build number.
///
/// - Returns: Version string from the app's Info.plist
var currentAppVersion: String {
    let version = Bundle.main.infoDictionary?["CFBundleShortVersionString"] as? String ?? "1.0"
    let build = Bundle.main.infoDictionary?["CFBundleVersion"] as? String ?? "1"
    return "\(version) (\(build))"
}
```

Property with Get/Set Documentation

swift

```
/// Controls whether the task list is in reordering mode
///
/// When `true`, drag handles appear on each task row allowing the user
/// to reorder tasks. When `false`, tasks are in normal interaction mode.
///
/// - Note: Setting this to `true` triggers haptic feedback
var isEditingTasks: Bool {
    get { _isEditingTasks }
    set {
        _isEditingTasks = newValue
        if newValue {
            #if os(iOS)
            HapticManager.shared.impact(style: .light)
            #endif
        }
    }
}
```

Documenting Enums

Basic Enum Documentation

swift

```swift
/// Represents the status of a task
enum TaskStatus: String, Codable {
    /// Task is not started
    case normal

    /// Task is currently being worked on
    case inProgress

    /// Task has been completed successfully
    case complete

    /// Task was not completed by deadline
    case notCompleted
}
```

Comprehensive Enum Documentation

```
swift
```

```swift
/// Represents the status of a task
///
/// Tasks can progress through different statuses as users work on them.
/// The status affects visual presentation and filtering in the UI.
///
/// ## Status Progression
/// The typical progression is:
/// ```
/// normal → inProgress → complete
///               ↘ notCompleted
/// ```
///
/// - Note: The default status for new tasks is `.normal`
/// - Important: Changing status triggers save to SwiftData
enum TaskStatus: String, Codable {
    /// Task has not been started
    ///
    /// This is the default status for newly created tasks.
    /// No visual indicator is shown in the UI.
    case normal

    /// Task is currently being worked on
    ///
    /// Displayed with a green clock icon in the task list.
    /// Use this to indicate active work in progress.
    case inProgress

    /// Task has been completed successfully
    ///
    /// Displayed with a green checkmark. Task name is struck through.
    /// This is the desired end state for most tasks.
    case complete
```

```
    /// Task was not completed by deadline
    ///
    /// Displayed with a red X icon. Indicates the task was
    /// abandoned or missed its deadline.
    case notCompleted
}
```

Enum with Associated Values

```
swift
```

```
/// Represents errors that can occur during CloudKit operations
///
/// These errors provide context about what went wrong during
/// CloudKit share creation, acceptance, or management.
enum CloudKitError: LocalizedError {
    /// Share creation failed
    ///
    /// - Parameter reason: Human-readable description of why creation failed
    case shareCreationFailed(reason: String)

    /// Record was not found in CloudKit
    ///
    /// This typically means the record hasn't synced yet.
    /// - SeeAlso: ``recordNotFoundAfterRetry``
    case recordNotFound

    /// Record still not found after retry attempts
    ///
    /// The record doesn't exist even after waiting and retrying.
    /// - Precondition: Must wait at least 10 seconds after creating record
    case recordNotFoundAfterRetry

    /// User is not signed into iCloud
    ///
    /// - Important: Users must be signed in to use CloudKit features
    case notSignedIntoiCloud

    var errorDescription: String? {
        switch self {
        case .shareCreationFailed(let reason):
            return "Failed to create share: \(reason)"
        case .recordNotFound:
            return "Task record not found in CloudKit"
        case .recordNotFoundAfterRetry:
            return "Task not synced to CloudKit yet. Please wait a few seconds and try again."
        case .notSignedIntoiCloud:
            return "Please sign into iCloud in Settings"
        }
    }
}
```

Markdown Support

Documentation comments support full Markdown formatting:

Headers

swift

```
/// # Main Heading
/// ## Subheading
/// ### Smaller Heading
```

Text Formatting

swift

```
/// You can use **bold text**, *italic text*, and `inline code`.
///
/// You can also use ~~strikethrough~~ text.
```

Lists

swift

```
/// Unordered list:
/// - First item
/// - Second item
/// - Third item
///
/// Ordered list:
/// 1. First step
/// 2. Second step
/// 3. Third step
```

Code Blocks

swift

```
/// Here's an example:
/// ```swift
/// let example = "Some code"
/// print(example)
/// ```
```

Links

swift

```
/// See [Apple's Documentation](https://developer.apple.com) for more info.
///
/// You can also link to symbols: ``OtherClass`` or ``myFunction(_:)``.
```

Blockquotes

swift

```
/// > This is a blockquote
/// > It can span multiple lines
```

Tables

swift

```
/// | Column 1 | Column 2 | Column 3 |
/// |----------|----------|----------|
/// | Value 1  | Value 2  | Value 3  |
/// | Value 4  | Value 5  | Value 6  |
```

Complete Markdown Example

swift

```swift
/// # Advanced Features Guide
///
/// This function provides **powerful** capabilities for data processing.
///
/// ## Supported Operations
///
/// The function supports these operations:
/// - **Filtering**: Remove unwanted items
/// - **Mapping**: Transform data
/// - **Sorting**: Order results
///
/// ## Usage
///
/// Here's how to use it:
///
/// ```swift
/// let data = [1, 2, 3, 4, 5]
/// let result = process(data, operation: .filter { $0 > 2 })
/// // result = [3, 4, 5]
/// ```
///
/// ## Performance
///
/// | Operation | Complexity |
/// |-----------|------------|
/// | Filter    | O(n)       |
/// | Map       | O(n)       |
/// | Sort      | O(n log n) |
///
/// > **Important**: This function assumes input is already validated.
///
/// - Complexity: O(n) where n is the number of items
/// - SeeAlso: [Swift Documentation](https://swift.org)
```

```
func process<T>(_ items: [T], operation: Operation) -> [T] {
    // ...
}
```

Complete Real-World Examples

Example 1: Toast Modifier (Complete)

```
swift
```

```
// MARK: - Toast View

/// A notification banner that appears temporarily at the top of the screen
///
/// `ToastView` displays a message with an icon in a frosted glass container.
/// It's typically used for brief confirmations like "Email sent" or "Task completed".
///
/// The view automatically includes:
/// - Material background with blur effect
/// - Colored icon and border
/// - Smooth spring animation
/// - Shadow for depth
///
/// - Important: This view should be presented via the ``toast(isPresented:message:icon:color:duration:)`` modifier,
///   not instantiated directly.
///
/// ## Example Appearance
/// ```
/// ┌─────────────────────────────────────┐
/// │  ✓ Thank you for feedback! │
/// └─────────────────────────────────────┘
/// ```
struct ToastView: View {
    /// The message text to display
    let message: String

    /// SF Symbol name for the icon
    let icon: String

    /// The accent color for the icon and border
    let color: Color

    var body: some View {
```

```
        HStack(spacing: 12) {
            Image(systemName: icon)
                .font(.title3)
                .foregroundColor(color)

            Text(message)
                .font(.subheadline)
                .fontWeight(.medium)
        }
        .padding(.horizontal, 16)
        .padding(.vertical, 12)
        .background(
            RoundedRectangle(cornerRadius: 12)
                .fill(.ultraThinMaterial)
                .shadow(color: .black.opacity(0.1), radius: 8, x: 0, y: 4)
        )
        .overlay(
            RoundedRectangle(cornerRadius: 12)
                .strokeBorder(color.opacity(0.3), lineWidth: 1)
        )
    }
}

// MARK: - View Extension

extension View {
    /// Presents a toast notification overlay
    ///
    /// Shows a temporary message banner that slides in from the top of the screen,
    /// automatically dismisses after the specified duration, and includes haptic feedback.
    ///
    /// - Parameters:
    ///   - isPresented: When `true`, the toast appears. Automatically resets to `false` after dismissal.
    ///   - message: The text to display in the toast notification
    ///   - icon: SF Symbol name for the leading icon (default: `"checkmark.circle.fill"`)
    ///   - color: Accent color for the icon and border (default: `.green`)
    ///   - duration: Time in seconds before auto-dismiss (default: `2.0`)
    ///
    /// - Returns: A view with the toast overlay modifier applied
    ///
    /// ## Usage
    /// Add the modifier to your view and control presentation with a binding:
    ///
    /// ```swift
    /// struct ContentView: View {
    ///     @State private var showToast = false
    ///
    ///     var body: some View {
    ///         Button("Save") {
    ///             // Perform save...
    ///             showToast = true
    ///         }
    ///         .toast(
    ///             isPresented: $showToast,
    ///             message: "Saved successfully!",
    ///             icon: "checkmark.circle.fill",
    ///             color: .green
    ///         )
    ///     }
    /// }
    /// ```
    ///
    /// ## Common Patterns
    ///
    /// **Success notification:**
    /// ```swift
```

```swift
/// .toast(isPresented: $showSuccess, message: "Done!", color: .green)
/// ```
///
/// **Error notification:**
/// ```swift
/// .toast(
///     isPresented: $showError,
///     message: "Something went wrong",
///     icon: "xmark.circle.fill",
///     color: .red
/// )
/// ```
///
/// **Info notification:**
/// ```swift
/// .toast(
///     isPresented: $showInfo,
///     message: "Syncing...",
///     icon: "arrow.triangle.2.circlepath",
///     color: .blue
/// )
/// ```
///
/// - Note: On iOS devices, the toast triggers success haptic feedback
/// - Tip: Keep messages concise (under 50 characters) for best readability
/// - Warning: Avoid showing multiple toasts simultaneously as they will overlap
func toast(
    isPresented: Binding<Bool>,
    message: String,
    icon: String = "checkmark.circle.fill",
    color: Color = .green,
    duration: TimeInterval = 2.0
) -> some View {
```

```
        modifier(ToastModifier(
            isPresented: isPresented,
            message: message,
            icon: icon,
            color: color,
            duration: duration
        ))
    }
}
```

Example 2: CloudKit Manager

```
swift
```

```
/// Manages CloudKit sharing operations for tasks
///
/// This class provides a high-level interface for creating, managing, and accepting
/// CloudKit shares for TaskItem objects. It coordinates between CloudKit, SwiftData,
/// and the UI to provide seamless task sharing functionality.
///
/// ## Features
/// - Create shares for individual tasks
/// - Update TaskItem properties when shares are created
/// - Accept incoming shares from other users
/// - Stop sharing and clean up shares
/// - Handle cancellation of share creation
///
/// ## Architecture
///
/// The manager acts as a coordinator between several components:
/// ```
/// UI → CloudKitSharingManager → CloudKitHelper → CloudKit
///                     ↓
///                 SwiftData
/// ```
///
/// ## Usage
///
/// ### Setup
/// Always inject a ModelContext before using:
/// ```swift
/// let manager = CloudKitSharingManager.shared
/// manager.modelContext = modelContext
/// ```
///
/// ### Creating a Share
/// ```swift
```

```
/// manager.shareTask(taskId, taskName: "My Task") { result in
///     switch result {
///     case .success(let (share, record)):
///         // Present share UI
///         self.activeShare = share
///     case .failure(let error):
///         // Handle error
///         print("Error: \(error)")
///     }
/// }
/// ```
///
/// ### Accepting a Share
/// ```swift
/// do {
///     try await manager.acceptShare(metadata: metadata)
///     print("Share accepted!")
/// } catch {
///     print("Failed to accept: \(error)")
/// }
/// ```
///
/// ## Topics
///
/// ### Creating and Managing Shares
/// - ``shareTask(_:taskName:completion:)``
/// - ``stopSharing(taskId:completion:)``
/// - ``cancelShare(taskId:share:completion:)``
///
/// ### Accepting Shares
/// - ``acceptShare(metadata:)``
///
/// ### State Properties
```

```
/// - ``activeShare``
/// - ``activeRecord``
/// - ``isLoading``
/// - ``sharingError``
///
/// ### Configuration
/// - ``modelContext``
///
/// - Important: This class must be used on the main actor
/// - Note: Always check ``isLoading`` before initiating new operations
/// - SeeAlso: ``CloudKitHelper``, ``SharePropertyUpdater``
@MainActor
class CloudKitSharingManager: ObservableObject {

    /// Shared singleton instance
    ///
    /// Use this shared instance throughout your app to maintain consistent state.
    ///
    /// - Important: Must set ``modelContext`` before use
    static let shared = CloudKitSharingManager()

    /// The CloudKit helper that performs low-level operations
    private let helper = CloudKitHelper.shared

    /// The currently active share being presented to the user
    ///
    /// Set when a share is successfully created or fetched. Used by the UI
    /// to present share options. Automatically cleared after sharing completes.
    @Published var activeShare: CKShare?

    /// The CKRecord associated with the active share
    ///
    /// Contains the actual task data in CloudKit format.
```

```swift
    @Published var activeRecord: CKRecord?

    /// Any error that occurred during sharing operations
    ///
    /// Check this property to display error messages to users.
    @Published var sharingError: Error?

    /// Whether a sharing operation is currently in progress
    ///
    /// Use this to show loading indicators in the UI.
    @Published var isLoading = false

    /// The SwiftData model context for database operations
    ///
    /// Must be set before using any sharing methods. Used to update
    /// TaskItem properties when shares are created or accepted.
    ///
    /// - Important: Always inject this on app startup
    var modelContext: ModelContext?

    // ... implementation
}
```

Example 3: Utility Function

```
swift
```

```swift
/// Formats a date relative to today
///
/// Returns human-readable strings like "Today", "Yesterday", "Tomorrow",
/// or the full date for dates further away.
///
/// - Parameter date: The date to format
/// - Returns: Formatted string representing the date
///
/// ## Examples
/// ```swift
/// formatRelativeDate(Date())          // "Today"
/// formatRelativeDate(Date().addingDays(-1))  // "Yesterday"
/// formatRelativeDate(Date().addingDays(1))   // "Tomorrow"
/// formatRelativeDate(Date().addingDays(-7))  // "Dec 2, 2024"
/// ```
///
/// - Note: Uses the current calendar and locale
/// - Complexity: O(1)
func formatRelativeDate(_ date: Date) -> String {
    let calendar = Calendar.current

    if calendar.isDateInToday(date) {
        return "Today"
    } else if calendar.isDateInYesterday(date) {
        return "Yesterday"
    } else if calendar.isDateInTomorrow(date) {
        return "Tomorrow"
    } else {
        let formatter = DateFormatter()
        formatter.dateStyle = .medium
        formatter.timeStyle = .none
        return formatter.string(from: date)
```

```
    }
}
```

Viewing Documentation

There are several ways to view documentation in Xcode:

1. Quick Help Popover

Keyboard: Option + Click on any symbol
Result: Shows a popover with documentation

2. Quick Help Inspector

Keyboard: ⌥⌘2 (Option + Command + 2)
Menu: View → Inspectors → Quick Help
Result: Shows documentation in the right sidebar as you navigate code

3. Documentation Window

Keyboard: ⌥⌘0 (Option + Command + 0)
Menu: Window → Developer Documentation
Result: Opens full documentation browser

4. Jump to Definition

Keyboard: Command + Click on symbol
Result: Jumps to the definition with full documentation visible

---

Best Practices

✅ Do

1. **Write the summary first**

```swift
/// Calculates the average of numbers  ← Start with this
///
/// Detailed description...
```

2. **Use present tense**

```swift
/// Saves the task to the database  ← Good
/// Will save the task            ← Bad
```

3. **Document all public APIs**
   - Every public function, class, property, and enum
   - Private code can skip documentation if obvious

4. **Include examples for complex APIs**

```swift
/// ## Example
/// ```swift
/// let result = complexFunction(param1, param2)
/// ```
```

5. **Keep it concise**
    - One-line summary (under 100 characters)
    - Detailed description if needed
    - Examples for complex cases
6. **Use proper capitalization and punctuation**

```swift
/// Loads user data from the server.  ← Good
/// loads user data                 ← Bad
```

7. **Update docs when code changes**
    - Outdated documentation is worse than no documentation
8. **Use semantic callouts appropriately**

```swift
/// - Important: Must be called on main thread
/// - Note: This may take several seconds
/// - Warning: Do not call while loading
```

❌ Don't

1. **Over-document obvious things**

```swift
/// Sets the name
/// - Parameter name: The name to set
var name: String  ← Too obvious, skip it
```

2. **Repeat information from the signature**

```swift
/// Function that takes a string and returns a string
func process(_ input: String) -> String  ← Signature already says this
```

3. **Write novels**
    - Keep it focused and relevant
    - Link to external docs for deep dives

4. **Use future tense**

```swift
/// Will process the data  ← Bad
/// Processes the data     ← Good
```

5. **Forget to update**

```swift
/// - Parameter oldName: The old parameter that no longer exists  ← Bad
```

6. **Mix documentation and regular comments**

```swift
/// This is documentation
// This is a regular comment  ← Don't mix in documentation
```

Quick Reference Templates

Function Template

swift

```swift
/// <One line summary>
///
/// <Detailed description if needed>
///
/// - Parameters:
///   - param1: <Description>
///   - param2: <Description>
/// - Returns: <What is returned>
/// - Throws: <What errors can be thrown>
///
/// ## Example
/// ```swift
/// let result = function(param1: value1, param2: value2)
/// ```
///
/// - Note: <Additional information>
/// - Important: <Critical information>
func functionName(param1: Type, param2: Type) throws -> ReturnType {
    // Implementation
}
```

Class/Struct Template

swift

```
/// <One line summary>
///
/// <Detailed description of purpose and functionality>
///
/// ## Overview
/// <High-level explanation of what this class does>
///
/// ## Topics
/// ### <Category Name>
/// - ``member1``
/// - ``member2``
///
/// ## Usage
/// ```swift
/// let instance = ClassName()
/// instance.doSomething()
/// ```
///
/// - Important: <Critical setup or usage information>
/// - Note: <Additional context>
/// - SeeAlso: ``RelatedClass``
class ClassName {
    // Implementation
}
```

Property Template

```
swift
```

```
/// <One line description>
///
/// <Detailed description including:
/// - What it represents
/// - When it changes
/// - What changes it
/// - Side effects>
///
/// - Note: <Additional information>
/// - Important: <Critical information about usage>
var propertyName: Type
```

Enum Template

```swift
/// <One line description of what this enum represents>
///
/// <Detailed description of purpose and usage>
///
/// - Note: <Additional context>
enum EnumName: RawType {
    /// <Description of this case>
    ///
    /// <When to use this case>
    case caseName

    /// <Description of this case>
    case anotherCase
}
```

Extension Template

```swift
// MARK: - <Category Name>

extension TypeName {
    /// <Description of what this extension adds>
    ///
    /// <Additional context about the extension's purpose>

    /// <Function documentation>
    func extensionFunction() {
        // Implementation
    }
}
```

Advanced Tips

Linking to Other Symbols

Use double backticks to create links to other types:

```swift
/// This function uses ``OtherClass`` and returns a ``ResultType``
///
/// - SeeAlso: ``relatedFunction(_:)``
func myFunction() -> ResultType
```

Symbol Resolution

Swift will auto-resolve these references:

- `TypeName` - Links to type
- `functionName(_:)` - Links to function (use `_:` for parameters)
- `propertyName` - Links to property
- `TypeName/memberName` - Links to member of type

Organizing with Topics

For complex types, use Topics to group members:

```swift
/// ## Topics
/// ### Creating Instances
/// - ``init()``
/// - ``init(name:)``
///
/// ### Properties
/// - ``name``
/// - ``age``
///
/// ### Methods
/// - ``save()``
/// - ``delete()``
```

Code Listings

For longer code examples, use fenced code blocks:

```swift
```

```
/// ## Example
/// ```swift
/// // Setup
/// let manager = Manager()
/// manager.configure()
///
/// // Usage
/// let result = manager.process(data)
/// print(result)
/// ```
```

Callout Stacking

You can use multiple callouts:

```swift
/// - Important: Must be called on main thread
/// - Note: This may take several seconds
/// - Warning: Do not call during initialization
/// - SeeAlso: ``alternativeMethod()``
```

Conclusion

Good documentation:

- Helps other developers (and future you) understand your code
- Makes your code more maintainable
- Provides context that code alone cannot

- Shows up beautifully in Xcode's Quick Help
- Makes your framework/library more professional

Remember:

- Start with a concise one-line summary
- Add details only where needed
- Use examples for complex cases
- Keep it updated as code changes
- Use proper grammar and formatting

Now go document your code! 📚 ✨

Resources

- Apple's API Design Guidelines
- Swift Documentation Markup
- Markdown Syntax

Version: 1.0
Last Updated: December 2024
Author: Swift Documentation Guide