Protocol

Scene

A part of an app’s user interface with a life cycle managed by the system.

iOS 14.0+ | iPadOS 14.0+ | Mac Catalyst 14.0+ | macOS 11.0+ | tvOS 14.0+ | visionOS 1.0+ | watchOS 7.0+

```
@MainActor @preconcurrency
protocol Scene
```

Mentioned in

- Building and customizing the menu bar with SwiftUI
- Migrating to the SwiftUI life cycle

Overview

You create an App by combining one or more instances that conform to the `Scene` protocol in the app’s body. You can use the built-in scenes that SwiftUI provides, like `WindowGroup`, along with custom scenes that you compose from other scenes. To create a custom scene, declare a type that conforms to the `Scene` protocol. Implement the required body computed property and provide the content for your custom scene:

```
struct MyScene: Scene {
    var body: some Scene {
        WindowGroup {
            MyRootView()
        }
    }
}
```

A scene acts as a container for a view hierarchy that you want to display to the user. The system decides when and how to present the view hierarchy in the user interface in a way that's platform-appropriate and dependent on the current state of the app. For example, for the window group shown above, the system lets the user create or remove windows that contain `MyRootView` on platforms like macOS and iPadOS. On other platforms, the same view hierarchy might consume the entire display when active.

Read the `scenePhase` environment value from within a scene or one of its views to check whether a scene is active or in some other state. You can create a property that contains the scene phase, which is one of the values in the `ScenePhase` enumeration, using the `Environment` attribute:

```
struct MyScene: Scene {
    @Environment(\.scenePhase) private var scenePhase

    // ...
}
```

The `Scene` protocol provides scene modifiers, defined as protocol methods with default implementations, that you use to configure a scene. For example, you can use the `onChange(of:perform:)` modifier to trigger an action when a value changes. The following code empties a cache when all of the scenes in the window group have moved to the background:

```
struct MyScene: Scene {
    @Environment(\.scenePhase) private var scenePhase
    @StateObject private var cache = DataCache()

    var body: some Scene {
        WindowGroup {
            MyRootView()
        }
        .onChange(of: scenePhase) { newScenePhase in
            if newScenePhase == .background {
                cache.empty()
            }
        }
    }
}
```

A type conforming to this protocol inherits `@preconcurrency @MainActor` isolation from the protocol if the conformance is included in the type's base declaration:

```
struct MyCustomType: Transition {
    // `@preconcurrency @MainActor` isolation by default
}
```

Isolation to the main actor is the default, but it’s not required. Declare the conformance in an extension to opt out of main actor isolation:

```
extension MyCustomType: Transition {
    // `nonisolated` by default
}
```

Topics

Creating a scene

`var body: Self.Body`

The content and behavior of the scene.

Required

`associatedtype Body : Scene`

The type of scene that represents the body of this scene.

Required

Watching for changes

`func onChange(of:initial:_:)`

Adds an action to perform when the given value changes.

`func handlesExternalEvents(matching: Set<String>) -> some Scene`

Specifies the external events for which SwiftUI opens a new instance of the modified scene.

Creating background tasks

`func backgroundTask<D, R>(BackgroundTask<D, R>, action: (D) async -> R) -> some Scene`

Runs the specified action when the system provides a background task.

Managing app storage

`func defaultAppStorage(UserDefaults) -> some Scene`

The default store used by `AppStorage` contained within the scene and its view content.

Setting commands

`func commands<Content>(content: () -> Content) -> some Scene`

Adds commands to the scene.

`func commandsRemoved() -> some Scene`

Removes all commands defined by the modified scene.

`func commandsReplaced<Content>(content: () -> Content) -> some Scene`

Replaces all commands defined by the modified scene with the commands from the builder.

`func keyboardShortcut(KeyboardShortcut?) -> some Scene`

Defines a keyboard shortcut for opening new scene windows.

`func keyboardShortcut(KeyEquivalent, modifiers: EventModifiers, localization: KeyboardShortcut.Localization) -> some Scene`

Defines a keyboard shortcut for opening new scene windows.

Sizing and positioning the scene

`func defaultPosition(UnitPoint) -> some Scene`

Sets a default position for a window.

`func defaultSize(_:)`

Sets a default size for a window.

`func defaultSize(width: CGFloat, height: CGFloat) -> some Scene`

Sets a default width and height for a window.

`func defaultSize(width: CGFloat, height: CGFloat, depth: CGFloat) -> some Scene`

Sets a default size for a volumetric window.

`func defaultSize(Size3D, in: UnitLength) -> some Scene`

Sets a default size for a volumetric window.

`func defaultSize(width: CGFloat, height: CGFloat, depth: CGFloat, in: UnitLength) -> some Scene`

Sets a default size for a volumetric window.

`func defaultWindowPlacement((WindowLayoutRoot, WindowPlacementContext) -> WindowPlacement) -> some Scene`

Defines a function used for determining the default placement of windows.

`func windowResizability(WindowResizability) -> some Scene`

Sets the kind of resizability to use for a window.

`func windowIdealSize(WindowIdealSize) -> some Scene`

Specifies how windows derived form this scene should determine their size when zooming.

`func windowIdealPlacement((WindowLayoutRoot, WindowPlacementContext) -> WindowPlacement) -> some Scene`

Provides a function which determines a placement to use when windows of a scene zoom.

`func windowManagerRole(WindowManagerRole) -> some Scene`

Configures the role for windows derived from `self` when participating in a managed window context, such as full screen or Stage Manager.

Interacting with volumes

`func volumeWorldAlignment(WorldAlignmentBehavior) -> some Scene`

Specifies how a volume should be aligned when moved in the world.

`func defaultWorldScaling(WorldScalingBehavior) -> some Scene`

Specify the world scaling behavior for the window.

Configuring scene visibility

`func defaultLaunchBehavior(SceneLaunchBehavior) -> some Scene`

Sets the default launch behavior for this scene.

`func restorationBehavior(SceneRestorationBehavior) -> some Scene`

Sets the restoration behavior for this scene.

`func persistentSystemOverlays(Visibility) -> some Scene`

Sets the preferred visibility of the non-transient system views overlaying the app.

Styling the scene

`func immersionStyle(selection: Binding<any ImmersionStyle>, in: any ImmersionStyle...) -> some Scene`

Sets the style for an immersive space.

`func upperLimbVisibility(Visibility) -> some Scene`

Sets the preferred visibility of the user's upper limbs, while an `ImmersiveSpace` scene is presented.

`func windowStyle<S>(S) -> some Scene`

Sets the style for windows created by this scene.

`func windowLevel(WindowLevel) -> some Scene`

Sets the window level of this scene.

`func windowToolbarStyle<S>(S) -> some Scene`

Sets the style for the toolbar defined within this scene.

`func windowToolbarLabelStyle(Binding<ToolbarLabelStyle>) -> some Scene`

Sets the label style of items in a toolbar and enables user customization.

`func windowToolbarLabelStyle(fixed: ToolbarLabelStyle) -> some Scene`

Sets the label style of items in a toolbar.

Configuring a data model

`func modelContext(ModelContext) -> some Scene`

Sets the model context in this scene's environment.

`func modelContainer(ModelContainer) -> some Scene`

Sets the model container and associated model context in this scene's environment.

`func modelContainer(for:inMemory:isAutosaveEnabled:isUndoEnabled:onSetup:)`

Sets the model container in this scene for storing the provided model type, creating a new container if necessary, and also sets a model context for that container in this scene's environment.

Managing the environment

`func environment<T>(T?) -> some Scene`

Places an observable object in the scene's environment.

`func environment<V>(WritableKeyPath<EnvironmentValues, V>, V) -> some Scene`

Sets the environment value of the specified key path to the given value.

`func environmentObject<T>(T) -> some Scene`

Supplies an `ObservableObject` to a view subhierarchy.

`func transformEnvironment<V>(WritableKeyPath<EnvironmentValues, V>, transform: (inout V) -> Void) -> some Scene`

Transforms the environment value of the specified key path with the given function.

Interacting with dialogs

`func dialogIcon(Image?) -> some Scene`

Configures the icon used by alerts.

`func dialogSeverity(DialogSeverity) -> some Scene`

Sets the severity for alerts.

`func dialogSuppressionToggle(isSuppressed: Binding<Bool>) -> some Scene`

Enables user suppression of an alert with a custom suppression message.

`func dialogSuppressionToggle(_:isSuppressed:)`

Enables user suppression of an alert with a custom suppression message.

Supporting drag behavior

`func windowBackgroundDragBehavior(WindowInteractionBehavior) -> some Scene`

Configures the behavior of dragging a window by its background.

Deprecated symbols

~~`func onChange<V>(of: V, perform: (V) -> Void) -> some Scene`~~

Adds an action to perform when the given value changes.

Deprecated

Instance Methods

`func documentBrowserContextMenu(([URL]?) -> some View) -> some Scene`

Adds to a `DocumentGroupLaunchScene` actions that accept a list of selected files as their parameter.

`func immersiveContentBrightness(ImmersiveContentBrightness) -> some Scene`

Sets the content brightness of an immersive space.

`func immersiveEnvironmentBehavior(ImmersiveEnvironmentBehavior) -> some Scene`

Sets the immersive environment behavior that should apply when this scene opens.

`func menuBarExtraStyle<S>(S) -> some Scene`

Sets the style for menu bar extra created by this scene.

Relationships

Conforming Types

`AlertScene`
`AssistiveAccess`
`DocumentGroup`
`DocumentGroupLaunchScene`
`Group`

Conforms when `Content` conforms to `Scene`.

`ImmersiveSpace`
`MenuBarExtra`
`ModifiedContent`

Conforms when `Content` conforms to `Scene` and `Modifier` conforms to `_SceneModifier`.

`RemoteImmersiveSpace`
`Settings`
`UtilityWindow`
`WKNotificationScene`
`Window`
`WindowGroup`

See Also

Creating scenes

`struct SceneBuilder`

A result builder for composing a collection of scenes into a single composite scene.