```
import CoreGraphics
import
DeveloperToolsSupport.DeveloperToolsSuppo
rt
import Foundation
import _Concurrency
import _StringProcessing
import _SwiftConcurrencyShims

/// A color resource.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct ColorResource : Hashable,
Sendable {

    /// Creates a color from a resource
with the specified name in the
    /// given bundle.
    public init(name: String, bundle:
Bundle)

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
```

```
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
    public static func == (a:
ColorResource, b: ColorResource) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
```

```
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// An image resource.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct ImageResource : Hashable,
Sendable {

    /// Creates an image from a resource
with the specified name in the
    /// given bundle.
    public init(name: String, bundle:
Bundle)

    /// Hashes the essential components
of this value by feeding them into the
    /// given hasher.
    ///
    /// Implement this method to conform
to the `Hashable` protocol. The
```

```
    /// components used for hashing must
be the same as the components compared
    /// in your type's `==` operator
implementation. Call `hasher.combine(_:)`
    /// with each of these components.
    ///
    /// - Important: In your
implementation of `hash(into:)`,
    ///   don't call `finalize()` on the
`hasher` instance provided,
    ///   or replace it with a different
instance.
    ///   Doing so may become a compile-
time error in the future.
    ///
    /// - Parameter hasher: The hasher to
use when combining the components
    ///   of this instance.
    public func hash(into hasher: inout
Hasher)

    /// Returns a Boolean value
indicating whether two values are equal.
    ///
    /// Equality is the inverse of
inequality. For any values `a` and `b`,
    /// `a == b` implies that `a != b` is
`false`.
    ///
    /// - Parameters:
    ///   - lhs: A value to compare.
    ///   - rhs: Another value to
compare.
```

```
    public static func == (a:
ImageResource, b: ImageResource) -> Bool

    /// The hash value.
    ///
    /// Hash values are not guaranteed to
be equal across different executions of
    /// your program. Do not save hash
values to use during a future execution.
    ///
    /// - Important: `hashValue` is
deprecated as a `Hashable` requirement.
To
    ///   conform to `Hashable`,
implement the `hash(into:)` requirement
instead.
    ///   The compiler provides an
implementation for `hashValue` for you.
    public var hashValue: Int { get }
}

/// A function builder for generating
arrays of library items without
/// requiring full array literal syntax.
///
/// Use the library content function
builder to simplify the implementation of
/// protocol requirements from you which
provide arrays of library
/// items. For example, without the
builder, you would have to explicitly put
/// items in an array in a
``LibraryContentProvider/views-25pdm``
```

```
/// implementation:
///
///     struct LibraryViewContent:
LibraryContentProvider {
///         var views: [LibraryItem] {
///             [
///
LibraryItem(MyFirstView()),
///
LibraryItem(MySecondView())
///             ]
///         }
///     }
///
/// With the builder, you can omit the
array literal syntax:
///
///     struct LibraryViewContent:
LibraryContentProvider {
///         @LibraryContentBuilder
///         var views: [LibraryItem] {
///
LibraryItem(MyFirstView())
///
LibraryItem(MySecondView())
///         }
///     }
///
/// In practice, the Swift compiler
infers the need for a library content
/// builder attribute and adds it at
build time, so that you never
/// need to explicitly write the
```

```
attribute in your code, even though it's
/// technically in use:
///
///     struct LibraryViewContent:
LibraryContentProvider {
///         var views: [LibraryItems] {
///
LibraryItem(MyFirstView())
///
LibraryItem(MySecondView())
///         }
///     }
///
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, visionOS 1.0, *)
@resultBuilder public struct
LibraryContentBuilder {

    public static func buildBlock(_
segments: [LibraryItem]...) ->
[LibraryItem]

    public static func buildExpression(_
item: LibraryItem) -> [LibraryItem]

    public static func buildExpression(_
items: [LibraryItem]) -> [LibraryItem]
}

/// A source of Xcode library and code
completion content.
///
/// Xcode discovers implementations of
```

```
the `LibraryContentProvider` protocol in
/// your project or workspace and
examines their contents for items that it
can
/// add to the Xcode library. Add views
by implementing the content provider's
/// computed
``DeveloperToolsSupport/LibraryContentPro
vider/views-25pdm``
/// property, and returning an array of
``DeveloperToolsSupport/LibraryItem``
/// instances initialized with the views
you want to publish:
///
///     struct LibraryViewContent:
LibraryContentProvider {
///         var views: [LibraryItem] {
///             LibraryItem(MyView())
///         }
///     }
///
/// Add view modifiers by implementing
the
///
``DeveloperToolsSupport/LibraryContentPro
vider/modifiers(base:)-4svii``
/// method and similarly returning an
array of library items initialized with
/// the modifiers you want to publish.
For view modifiers, you also specify the
/// type to which the modifiers apply:
///
///     struct LibraryModifierContent:
```

```
LibraryContentProvider {
///         func modifiers(base: MyView)
-> [LibraryItem] {
///
LibraryItem(base.myModifier(value:
MyValue()))
///         }
///     }
///
/// For modifiers that you define in an
extension to
///
<doc://com.apple.documentation/documentat
ion/SwiftUI/View>, you can provide
/// any view conformer as the `base`. For
modifiers that you define on a
/// particular view type, provide that
type as the `base`.
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, visionOS 1.0, *)
public protocol LibraryContentProvider {

    /// A type to use as a base for
modifier completions.
    ///
    /// To verify that the completion for
a modifier compiles, you specify
    /// modifiers as they apply to some
base type. Since most modifiers can
    /// modify any SwiftUI view, you
typically specify any concrete
    /// implementation of the
    ///
```

```
<doc://com.apple.documentation/documentation/SwiftUI/View> protocol for
    /// `ModifierBase`. However, some modifiers apply to more specific types,
    /// like <doc://com.apple.documentation/documentation/SwiftUI/Image> or
    /// <doc://com.apple.documentation/documentation/SwiftUI/Text>,
    /// or to an entirely different type like
    /// <doc://com.apple.documentation/documentation/SwiftUI/Shape>.
    associatedtype ModifierBase = Any

    /// The SwiftUI views that you want to add to the Xcode library.
    ///
    /// Xcode adds the ``DeveloperToolsSupport/LibraryItem`` instances returned
    /// by your implementation of this property to its Views library. The
    /// following restrictions apply:
    ///  - You must instantiate the library items inline.
    ///  - If specified, the item's `title`, `category`, and `matchingSignature`
    ///    must be static strings and not `nil`.
```

```
    ///  - The item's `visible` value, if
specified, must be a literal Boolean
    ///    value.
    ///  - The item's expression must be
an instantiation. That is, it can't be
    ///    a reference.
    @LibraryContentBuilder var views:
[LibraryItem] { get }

    /// Indicates a collection of SwiftUI
view modifiers to add to the Xcode
    /// library.
    ///
    /// Xcode adds the
``DeveloperToolsSupport/LibraryItem``
instances returned
    /// by your implementation of this
method to its Modifiers library. The
    /// following restrictions apply:
    ///  - You must instantiate the
library items inline.
    ///  - If specified, the item's
`title`, `category`, and
`matchingSignature`
    ///    must be static strings and
not `nil`.
    ///  - The item's `visible` value, if
specified, must be a literal Boolean
    ///    value.
    ///  - The item's expression must be
a reference expression where the root
    ///    reference is `base` and the
expression contains at least one
```

```
    ///     modifier, like
`base.opacity(0.5)`.
    ///
    /// - Parameters:
    ///   - base: An instance to apply
modifiers to when declaring a library
    ///     item.
    @LibraryContentBuilder func
modifiers(base: Self.ModifierBase) ->
[LibraryItem]
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, visionOS 1.0, *)
extension LibraryContentProvider {

    /// The SwiftUI views that you want
to add to the Xcode library.
    ///
    /// Xcode adds the
``DeveloperToolsSupport/LibraryItem``
instances returned
    /// by your implementation of this
property to its Views library. The
    /// following restrictions apply:
    ///  - You must instantiate the
library items inline.
    ///  - If specified, the item's
`title`, `category`, and
`matchingSignature`
    ///     must be static strings and
not `nil`.
    ///  - The item's `visible` value, if
```

```
specified, must be a literal Boolean
    ///     value.
    ///  - The item's expression must be
an instantiation. That is, it can't be
    ///     a reference.
    public var views: [LibraryItem] { get
}

    /// Indicates a collection of SwiftUI
view modifiers to add to the Xcode
    /// library.
    ///
    /// Xcode adds the
``DeveloperToolsSupport/LibraryItem``
instances returned
    /// by your implementation of this
method to its Modifiers library. The
    /// following restrictions apply:
    ///  - You must instantiate the
library items inline.
    ///  - If specified, the item's
`title`, `category`, and
`matchingSignature`
    ///     must be static strings and
not `nil`.
    ///  - The item's `visible` value, if
specified, must be a literal Boolean
    ///     value.
    ///  - The item's expression must be
a reference expression where the root
    ///     reference is `base` and the
expression contains at least one
    ///     modifier, like
```

```
`base.opacity(0.5)`.
    ///
    /// - Parameters:
    ///   - base: An instance to apply
modifiers to when declaring a library
    ///     item.
    public func modifiers(base:
Self.ModifierBase) -> [LibraryItem]
}

/// A single item to add to the Xcode
library.
///
/// Declare a library item to describe an
entry in the Xcode library.
/// Xcode discovers and validates library
items that you place in the context of
/// a
``DeveloperToolsSupport/LibraryContentPro
vider`` instance.
///
/// At a minimum, you provide an
expression that Xcode uses when the user
/// chooses the library item. You can
provide any expression that compiles in
/// the context of the library item
instantiation. However, Xcode only honors
/// items that adhere to certain
restrictions, as described in
///
``DeveloperToolsSupport/LibraryContentPro
vider/views-25pdm`` and
///
```

```
``DeveloperToolsSupport/LibraryContentProvider/modifiers(base:)-4svii``.
///
/// You can also provide additional characteristics, like a title
/// and a category, to help you find the item when searching the library.
@available(iOS 14.0, macOS 11.0, tvOS 14.0, watchOS 7.0, visionOS 1.0, *)
public struct LibraryItem {

    /// The kinds of library items that you can create.
    ///
    /// When you specify a category for a library item, Xcode can group it
    /// with similar items in the library, making it easier for you to find.
    /// Categories provide visual treatment in the Xcode Library, but the
    /// treatment for each category depends on where the asset resides within
    /// the library.
    @available(iOS 14.0, macOS 11.0, tvOS 14.0, watchOS 7.0, visionOS 1.0, *)
    public struct Category {

        /// A category for effects, like opacity and saturation modifiers.
        public static let effect: LibraryItem.Category
```

```
        /// A category for items that
manage layout, like stack views and frame
        /// modifiers.
        public static let layout:
LibraryItem.Category

        /// A category for controls, like
buttons and context menus.
        public static let control:
LibraryItem.Category

        /// A general category.
        public static let other:
LibraryItem.Category
    }

    /// Creates a new library item.
    /// - Parameters:
    ///   - snippet: The expression to
insert when the user picks the item
    ///     from the library, or inserts
it via code completion.
    ///   - visible: A Boolean that
specifies whether to make this item
visible in the
    ///     library. You might choose to
hide an item from the library if its only
    ///     purpose is to support code
completion.
    ///   - title: A title for the item
in the library.
    ///     If unspecified, Xcode
generates a default title.
```

```
    ///   - category: A category for the
item in the library.
    ///     If unspecified, Xcode assumes
the default category of "Other".
    ///   - matchingSignature: An
overload for which the item presents its
code
    ///     completion. You typically use
this parameter when setting up an item as
a
    ///     source of custom code
completion. At the time of completion,
the code
    ///     completion engine looks for
an item matching the signature and
    ///     inserts its completion, if
found.
    public init<SnippetExpressionType>(_
snippet: @autoclosure () ->
SnippetExpressionType, visible: Bool =
true, title: String? = nil, category:
LibraryItem.Category = .other,
matchingSignature: String? = nil)
}

/// A base type that preview macros use
to create previews.
///
/// Frameworks like SwiftUI and WidgetKit
define initializers for this type,
/// along with framework-specific preview
macros that rely on this type.
/// You don't use this type directly.
```

```
Instead, use one of the preview macros,
/// like
<doc://com.apple.documentation/documentat
ion/SwiftUI/Preview(_:body:)>.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@MainActor public struct Preview {
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Preview {

    /// Traits that apply to previews of
views and view controllers.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public enum ViewTraits {
    }
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Preview : Sendable {
}

/// Builder for preview body content
within a `#Preview` macro.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
@resultBuilder public struct
PreviewBodyBuilder<Content> {
```

```
    public static func buildBlock(_
content: Content) -> Content
}

/// A size constraint for a preview.
///
/// Customize the layout of a preview
that you define using the
///
<doc://com.apple.documentation/documentat
ion/SwiftUI/PreviewProvider>
/// protocol by providing one of the
preview layout values to the
///
<doc://com.apple.documentation/documentat
ion/SwiftUI/View/previewLayout(_:)>
/// view modifier. For example, you can
tell the preview to take up only the
/// amount of space that the view
requires with ``sizeThatFits``:
///
///     struct CircleImage_Previews:
PreviewProvider {
///         static var previews: some
View {
///             CircleImage()
///                 .previewLayout(.sizeT
hatFits)
///         }
///     }
///
/// > Note: When you migrate away from
preview providers and to preview macros,
```

```
///   you specify layout using one of the
``PreviewTrait`` layout
///   values with a macro that takes
traits, like
///
<doc://com.apple.documentation/documentat
ion/SwiftUI/Preview(_:traits:_:body:)>.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public enum PreviewLayout : Sendable {

    /// Center the preview in a container
the size of the device on which the
    /// preview is running.
    case device

    /// Fit the container to the size of
the preview when offered the size of
    /// the device that the preview is
running on.
    case sizeThatFits

    /// Center the preview in a fixed
size container with the given dimensions.
    ///
    /// - Parameters:
    ///   - width: The width of the
container.
    ///   - height: The height of the
container.
    case fixed(width: CGFloat, height:
CGFloat)
}
```

```
/// Builder for preview body content
within a `#Preview` macro.
///
/// - Warning: Using this builder outside
of a `#Preview` macro will produce a
fatal error.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, visionOS 1.0, *)
@resultBuilder public struct
PreviewMacroBodyBuilder<Content> {

    public static func buildBlock(_:
Content) -> Content
}

/// A protocol that the system uses to
locate previews at runtime.
///
/// Preview macros make use of this
protocol on your behalf.
/// Don't use it directly. Instead, use
one of the preview macros, like
///
<doc://com.apple.documentation/documentat
ion/SwiftUI/Preview(_:body:)>.
///
/// > Important: If you define a preview
registry directly, the behavior is
///   undefined.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public protocol PreviewRegistry {
```

```
    static var fileID: String { get }

    static var line: Int { get }

    static var column: Int { get }

    @MainActor static func makePreview()
throws -> Preview

    @available(*, deprecated, message:
"This method is not called. Please
implement makePreview() instead.")
    static var preview: Preview { get }
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension PreviewRegistry {

    public static var preview: Preview {
get }
}

/// Customizations that you can apply to
a preview.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@MainActor public struct PreviewTrait<T>
{

    /// Convenience to compose multiple
traits into a single trait.
```

```
    @available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    @MainActor public init(_ traits:
PreviewTrait<T>...)
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension PreviewTrait where T ==
Preview.ViewTraits {

    /// Center the preview in a container
the size of the device on which the
    /// preview is running.
    ///
    /// This is the same as the
``PreviewLayout/device`` layout, and is
the
    /// default if you don't specify a
layout trait.
    @MainActor public static var
defaultLayout:
PreviewTrait<Preview.ViewTraits> { get }

    /// Fit the container to the size of
the preview when offered the size of
    /// the device that the preview is
running on.
    ///
    /// This is the same as
``PreviewLayout/sizeThatFits``.
    @MainActor public static var
sizeThatFitsLayout:
```

```
PreviewTrait<Preview.ViewTraits> { get }

    /// Center the preview in a fixed
size container with the given dimensions.
    ///
    /// This is the same as
``PreviewLayout/fixed(width:height:)``.
    @MainActor public static func
fixedLayout(width: CGFloat, height:
CGFloat) -> PreviewTrait<T>
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension PreviewTrait where T ==
Preview.ViewTraits {

    /// The device is in portrait mode,
with the top of the device on top.
    ///
    /// This is the same as
    ///
<doc://com.apple.documentation/documentat
ion/SwiftUI/InterfaceOrientation/
portrait>
    /// and is the default orientation if
you don't specify one.
    @MainActor public static var
portrait:
PreviewTrait<Preview.ViewTraits> { get }

    /// The device is in landscape mode,
with the top of the device on the left.
```

```
    ///
    /// This is the same as
    ///
<doc://com.apple.documentation/documentat
ion/SwiftUI/InterfaceOrientation/
landscapeLeft>.
    @MainActor public static var
landscapeLeft:
PreviewTrait<Preview.ViewTraits> { get }

    /// The device is in landscape mode,
with the top of the device on the right.
    ///
    /// This is the same as
    ///
<doc://com.apple.documentation/documentat
ion/SwiftUI/InterfaceOrientation/
landscapeRight>.
    @MainActor public static var
landscapeRight:
PreviewTrait<Preview.ViewTraits> { get }

    /// The device is in portrait mode,
but is upside down.
    ///
    /// This is the same as
    ///
<doc://com.apple.documentation/documentat
ion/SwiftUI/InterfaceOrientation/
portraitUpsideDown>.
    @MainActor public static var
portraitUpsideDown:
PreviewTrait<Preview.ViewTraits> { get }
```

```
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension PreviewTrait : Sendable {
}

/// An error that the system throws when
a preview is unavailable at runtime.
///
/// This type supports the expansion of
preview macros. You don't use it
/// directly.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct PreviewUnavailable : Error
{

    public init()
}
```