

```
import Accessibility
import Combine
import CoreFoundation
import CoreGraphics
import CoreText
import CoreTransferable
import DeveloperToolsSupport
import Foundation
import Observation
import QuartzCore
import Symbols
import TargetConditionals
import UniformTypeIdentifiers
import _Concurrency
import _StringProcessing
import _SwiftConcurrencyShims
import os
import os.log
import simd
```

```
/// A type to generate an
`AXChartDescriptor` object that you use
to provide
/// information about a chart and its
data for an accessible experience
/// in VoiceOver or other assistive
technologies.
///
/// Note that you may use the
`@Environment` property wrapper inside
the
/// implementation of your
`AXChartDescriptorRepresentable`, in
```

```
which case you
/// should implement
`updateChartDescriptor`, which will be
called when the
/// `Environment` changes.
///
/// For example, to provide accessibility
for a view that represents a chart,
/// you would first declare your chart
descriptor representable type:
///
///     struct
MyChartDescriptorRepresentable: AXChartDescriptorRepresentable {
    ///         func makeChartDescriptor() ->
AXChartDescriptor {
    ///             // Build and return your
`AXChartDescriptor` here.
    ///         }
    ///
    ///         func updateChartDescriptor(_
descriptor: AXChartDescriptor) {
    ///             // Update your chart
descriptor with any new values.
    ///         }
    ///     }
    ///
    /// Then, provide an instance of your
`AXChartDescriptorRepresentable` type to
/// your view using the
`accessibilityChartDescriptor` modifier:
///
///     var body: some View {
```

```
///         MyChartView()
///         .accessibilityChartDescri
ptor(MyChartDescriptorRepresentable())
///     }
/// 
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
public protocol
AXChartDescriptorRepresentable {

    /// Create the `AXChartDescriptor`  

for this view, and return it.
    ///
    /// This will be called once per  

identity of your `View`. It will not be  

run
    /// again unless the identity of your  

`View` changes. If you need to
    /// update the `AXChartDescriptor`  

based on changes in your `View`, or in
    /// the `Environment`, implement
`updateChartDescriptor`.

    /// This method will only be called
if / when accessibility needs the
    /// `AXChartDescriptor` of your view,
for VoiceOver.
    func makeChartDescriptor() ->
AXChartDescriptor

    /// Update the existing
`AXChartDescriptor` for your view, based
on changes
    /// in your view or in the
```

```
`Environment`.  
    ///  
    /// This will be called as needed,  
when accessibility needs your  
    /// `AXChartDescriptor` for  
VoiceOver. It will only be called if the  
inputs  
    /// to your views, or a relevant part  
of the `Environment`, have changed.  
    func updateChartDescriptor(_  
descriptor: AXChartDescriptor)  
{  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension AXChartDescriptorRepresentable  
{  
  
    /// Update the existing  
`AXChartDescriptor` for your view, based  
on changes  
    /// in your view or in the  
`Environment`.  
    ///  
    /// This will be called as needed,  
when accessibility needs your  
    /// `AXChartDescriptor` for  
VoiceOver. It will only be called if the  
inputs  
    /// to your views, or a relevant part  
of the `Environment`, have changed.  
    public func updateChartDescriptor(_  
descriptor: AXChartDescriptor)
```

```
}
```

```
/// Key used to specify the identifier  
and label associated with  
/// an entry of additional accessibility  
information.  
///  
/// Use `AccessibilityCustomContentKey`  
and the associated modifiers taking  
/// this value as a parameter in order to  
simplify clearing or replacing  
/// entries of additional information  
that are manipulated from multiple places  
/// in your code.  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
public struct  
AccessibilityCustomContentKey {  
  
    /// Create an  
`AccessibilityCustomContentKey` with the  
specified label and  
    /// identifier.  
    ///  
    /// - Parameter label: Localized text  
describing to the user what  
    ///     is contained in this additional  
information entry. For example:  
    ///     "orientation".  
    /// - Parameter id: String used to  
identify the additional information entry  
    ///     to SwiftUI. Adding an entry  
will replace any previous value with the
```

```
    /// same identifier.  
    public init(_ label: Text, id:  
String)  
  
        /// Create an  
`AccessibilityCustomContentKey` with the  
specified label and  
        /// identifier.  
        ///  
        /// - Parameter labelKey: Localized  
text describing to the user what  
        /// is contained in this additional  
information entry. For example:  
        /// "orientation".  
        /// - Parameter id: String used to  
identify the additional  
        /// information entry to SwiftUI.  
Adding an entry will replace any previous  
        /// value with the same identifier.  
    public init(_ labelKey:  
LocalizedStringKey, id: String)  
  
        /// Create an  
`AccessibilityCustomContentKey` with the  
specified label.  
        ///  
        /// - Parameter labelKey: Localized  
text describing to the user what  
        /// is contained in this additional  
information entry. For example:  
        /// "orientation". This will also  
be used as the identifier.  
    public init(_ labelKey:
```

```
LocalizedStringRef)
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension AccessibilityCustomContentKey :  
Equatable {  
  
    /// 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:  
        AccessibilityCustomContentKey, b:  
        AccessibilityCustomContentKey) -> Bool  
}  
  
/// The hierarchy of a heading in  
relation other headings.  
///  
/// Assistive technologies can use this  
to improve a users navigation  
/// through multiple headings. When users  
navigate through top level  
/// headings they expect the content for
```

each heading to be unrelated.

///

/// For example, you can categorize a  
list of available products into sections,  
/// like Fruits and Vegetables. With only  
top level headings, this list requires no  
/// heading hierarchy, and you use the  
``unspecified`` heading level. On the

other hand, if sections

/// contain subsections, like if the  
Fruits section has subsections for  
varieties of Apples,

/// Pears, and so on, you apply the  
``h1`` level to Fruits and Vegetables,  
and the ``h2``

/// level to Apples and Pears.

///

/// Except for ``h1``, be sure to precede  
all leveled headings by another heading  
with a level

/// that's one less.

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)
```

```
@frozen public enum
```

```
AccessibilityHeadingLevel : UInt {
```

```
    /// A heading without a hierarchy.  
    case unspecified
```

```
    /// Level 1 heading.  
    case h1
```

```
    /// Level 2 heading.
```

```
case h2

    /// Level 3 heading.
case h3

    /// Level 4 heading.
case h4

    /// Level 5 heading.
case h5

    /// Level 6 heading.
case h6

    /// Creates a new instance with the
    specified raw value.
    ///
    /// If there is no value of the type
    that corresponds with the specified raw
    /// value, this initializer returns
    `nil`. For example:
    ///
    ///     enum PaperSize: String {
    ///         case A4, A5, Letter,
Legal
    ///     }
    ///
    ///     print(PaperSize(rawValue:
    "Legal"))
    ///     // Prints
    "Optional("PaperSize.Legal")"
    ///
    ///     print(PaperSize(rawValue:
```

```
"Tabloid"))
    /**
     // Prints "nil"
    /**
     /**
     - Parameter rawValue: The raw
     value to use for the new instance.
    public init?(rawValue: UInt)

    /**
     The raw type that can be used to
     represent all values of the conforming
     /**
     type.
    /**
    /**
     Every distinct value of the
     conforming type has a corresponding
     unique
    /**
     value of the `RawValue` type, but
     there may be values of the `RawValue`
     /**
     type that don't have a
     corresponding value of the conforming
     type.
    @available(iOS 15.0, tvOS 15.0,
    watchOS 8.0, macOS 12.0, *)
    public typealias RawValue = UInt

    /**
     The corresponding value of the
     raw type.
    /**
    /**
     A new instance initialized with
     `rawValue` will be equivalent to this
     /**
     instance. For example:
    /**
    /**
     enum PaperSize: String {
        /**
         case A4, A5, Letter,
     Legal
```

```
    /**
     */
    /**
     *      let selectedSize =
PaperSize.Letter
    /**
     *      print(selectedSize.rawValue)
    /**
     *      // Prints "Letter"
    /**
     *      print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    /**
     *      // Prints "true"
public var rawValue: UInt { get }
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension AccessibilityHeadingLevel : Equatable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension AccessibilityHeadingLevel : Hashable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension AccessibilityHeadingLevel : RawRepresentable {
}

@available(iOS 15.0, macOS 12.0, tvOS
```

```
15.0, watchOS 8.0, *)
extension AccessibilityHeadingLevel :  
Sendable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)
extension AccessibilityHeadingLevel :  
BitwiseCopyable {  
}  
  
/// Textual context that assistive  
technologies can use to improve the  
/// presentation of spoken text.  
///  
/// Use an `AccessibilityTextContentType`  
value when setting the accessibility text  
content  
/// type of a view using the  
``View/accessibilityTextContentType(_:)``  
modifier.  
///  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)
public struct  
AccessibilityTextContentType : Sendable {  
  
    /// A type that represents generic  
text that has no specific type.  
    public static let plain:  
AccessibilityTextContentType  
  
    /// A type that represents text used
```

for input, like in the Terminal app.

```
public static let console:  
AccessibilityTextContentType
```

// A type that represents text used by a file browser, like in the Finder app in macOS.

```
public static let fileSystem:  
AccessibilityTextContentType
```

// A type that represents text used in a message, like in the Messages app.

```
public static let messaging:  
AccessibilityTextContentType
```

// A type that represents text used in a story or poem, like in the Books app.

```
public static let narrative:  
AccessibilityTextContentType
```

// A type that represents text used in source code, like in Swift Playgrounds.

```
public static let sourceCode:  
AccessibilityTextContentType
```

// A type that represents text used in a grid of data, like in the Numbers app.

```
public static let spreadsheet:  
AccessibilityTextContentType
```

```
    /// A type that represents text used
    /// in a document, like in the Pages app.
    public static let wordProcessing:
AccessibilityTextContentType
}

/// A set of accessibility traits that
/// describe how an element behaves.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public struct AccessibilityTraits :  
SetAlgebra, Sendable {

    /// The accessibility element is a
    /// button.
    public static let isButton:
AccessibilityTraits

    /// The accessibility element is a
    /// header that divides content into
    /// sections, like the title of a
    /// navigation bar.
    public static let isHeader:
AccessibilityTraits

    /// The accessibility element is
    /// currently selected.
    public static let isSelected:
AccessibilityTraits

    /// The accessibility element is a
    /// link.
    public static let isLink:
```

## AccessibilityTraits

```
    /// The accessibility element is a  
    search field.
```

```
    public static let isSearchField:  
        AccessibilityTraits
```

```
    /// The accessibility element is an  
    image.
```

```
    public static let isImage:  
        AccessibilityTraits
```

```
    /// The accessibility element plays  
    its own sound when activated.
```

```
    public static let playsSound:  
        AccessibilityTraits
```

```
    /// The accessibility element behaves  
    as a keyboard key.
```

```
    public static let isKeyboardKey:  
        AccessibilityTraits
```

```
    /// The accessibility element is a  
    static text that cannot be
```

```
    /// modified by the user.
```

```
    public static let isStaticText:  
        AccessibilityTraits
```

```
    /// The accessibility element  
    provides summary information when the
```

```
    /// application starts.
```

```
    ///
```

```
    /// Use this trait to characterize an
```

```
accessibility element that provides
    /// a summary of current conditions,
settings, or state, like the
    /// temperature in the Weather app.
public static let isSummaryElement:
AccessibilityTraits
```

```
    /// The accessibility element
frequently updates its label or value.
    ///
    /// Use this trait when you want an
assistive technology to poll for
    /// changes when it needs updated
information. For example, you might use
    /// this trait to characterize the
readout of a stopwatch.
public static let updatesFrequently:
AccessibilityTraits
```

```
    /// The accessibility element starts
a media session when it is activated.
    ///
    /// Use this trait to silence the
audio output of an assistive technology,
    /// such as VoiceOver, during a media
session that should not be interrupted.
    /// For example, you might use this
trait to silence VoiceOver speech while
    /// the user is recording audio.
public static let startsMediaSession:
AccessibilityTraits
```

```
    /// The accessibility element allows
```

```
direct touch interaction for
    /// VoiceOver users.
    public static let
allowsDirectInteraction:
AccessibilityTraits
```

```
    /// The accessibility element causes
an automatic page turn when VoiceOver
    /// finishes reading the text within
it.
```

```
    public static let causesPageTurn:
AccessibilityTraits
```

```
    /// The accessibility element is
modal.
```

```
    ///
    /// Use this trait to restrict which
accessibility elements an assistive
    /// technology can navigate. When a
modal accessibility element is visible,
    /// sibling accessibility elements
that are not modal are ignored.
```

```
    public static let isModal:
AccessibilityTraits
```

```
    /// The accessibility element is a
toggle.
```

```
    @available(macOS 14.0, iOS 17.0, tvOS
17.0, watchOS 10.0, *)
    public static let isToggle:
AccessibilityTraits
```

```
    /// The accessibility element is a
```

```
tab bar
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public static let isTabBar:
AccessibilityTraits

    /// Creates an empty set.
    ///
    /// This initializer is equivalent to
    initializing with an empty array
    /// literal. For example, you create
    an empty `Set` instance with either
    /// this initializer or with an empty
    array literal.
    ///
    ///     var emptySet = Set<Int>()
    ///     print(emptySet.isEmpty)
    ///     // Prints "true"
    ///
    ///     emptySet = []
    ///     print(emptySet.isEmpty)
    ///     // Prints "true"
public init()

    /// Returns a new set with the
    elements of both this and the given set.
    ///
    /// In the following example, the
    `attendeesAndVisitors` set is made up
    /// of the elements of the
    `attendees` and `visitors` sets:
    ///
    ///     let attendees: Set =
```

```
["Alicia", "Bethany", "Diana"]
    ///      let visitors = ["Marcia",
"Nathaniel"]
    ///      let attendeesAndVisitors =
attendees.union(visitors)
    ///      print(attendeesAndVisitors)
    ///      // Prints "["Diana",
"Nathaniel", "Bethany", "Alicia",
"Marcia"]"
    ///
    /// If the set already contains one
or more elements that are also in
    /// `other`, the existing members are
kept.
    ///
    ///      let initialIndices =
Set(0..<5)
    ///      let expandedIndices =
initialIndices.union([2, 3, 6, 7])
    ///      print(expandedIndices)
    ///      // Prints "[2, 4, 6, 7, 0, 1,
3]"
    ///
    /// - Parameter other: A set of the
same type as the current set.
    /// - Returns: A new set with the
unique elements of this set and `other` .
    ///
    /// - Note: if this set and `other`
contain elements that are equal but
    /// distinguishable (e.g. via
`==`), which of these elements is
present
```

```
    /// in the result is unspecified.  
    public func union(_ other:  
        AccessibilityTraits) ->  
        AccessibilityTraits  
  
        /// Adds the elements of the given  
        set to the set.  
        ///  
        /// In the following example, the  
        elements of the `visitors` set are added  
        to  
        /// the `attendees` set:  
        ///  
        ///     var attendees: Set =  
        ["Alicia", "Bethany", "Diana"]  
        ///     let visitors: Set = ["Diana",  
        "Marcia", "Nathaniel"]  
        ///     attendees.formUnion(visitors)  
        ///     print(attendees)  
        ///     // Prints "["Diana",  
        "Nathaniel", "Bethany", "Alicia",  
        "Marcia"]"  
        ///  
        /// If the set already contains one  
        or more elements that are also in  
        /// `other`, the existing members are  
        kept.  
        ///  
        ///     var initialIndices =  
        Set(0..<5)  
        ///     initialIndices.formUnion([2,  
        3, 6, 7])  
        ///     print(initialIndices)
```

```
    ///      // Prints "[2, 4, 6, 7, 0, 1,
3]"
    ///
    /// - Parameter other: A set of the
same type as the current set.
public mutating func formUnion(_
other: AccessibilityTraits)

    /// Returns a new set with the
elements that are common to both this set
and
    /// the given set.
    ///
    /// In the following example, the
`bothNeighborsAndEmployees` set is made
up
    /// of the elements that are in
*both* the `employees` and `neighbors`
sets.
    /// Elements that are in only one or
the other are left out of the result of
    /// the intersection.
    ///
    ///     let employees: Set =
["Alicia", "Bethany", "Chris", "Diana",
"Eric"]
    ///     let neighbors: Set =
["Bethany", "Eric", "Forlani", "Greta"]
    ///     let bothNeighborsAndEmployees
= employees.intersection(neighbors)
    ///
print(bothNeighborsAndEmployees)
    ///     // Prints "["Bethany",
```

```
"Eric"]"
  /**
   /**
   -- Parameter other: A set of the
   same type as the current set.
   /**
   -- Returns: A new set.
   /**
   /**
   -- Note: if this set and `other`
   contain elements that are equal but
   /**
   -- distinguishable (e.g. via
   `==`), which of these elements is
   present
   /**
   -- in the result is unspecified.
public func intersection(_ other:
AccessibilityTraits) ->
AccessibilityTraits
```

```
 /**
   -- Removes the elements of this set
   that aren't also in the given set.
   /**
   /**
   -- In the following example, the
   elements of the `employees` set that are
   /**
   -- not also members of the
   `neighbors` set are removed. In
   particular, the
   /**
   -- names `"Alicia"`, `"Chris"`, and
   `"Diana"` are removed.
   /**
   /**
   --     var employees: Set =
["Alicia", "Bethany", "Chris", "Diana",
"Eric"]
   /**
   --     let neighbors: Set =
["Bethany", "Eric", "Forlani", "Greta"]
   /**/
```

```
employees.formIntersection(neighbors)
    ///      print(employees)
    ///      // Prints "[""Bethany"",
"Eric"]"
    ///
    /// - Parameter other: A set of the
    same type as the current set.
public mutating func
formIntersection(_ other:
AccessibilityTraits)

    /// Returns a new set with the
elements that are either in this set or
in the
    /// given set, but not in both.
    ///
    /// In the following example, the
`eitherNeighborsOrEmployees` set is made
up
    /// of the elements of the
`employees` and `neighbors` sets that are
not in
    /// both `employees` *and*
`neighbors`. In particular, the names
`"Bethany"`
    /// and `"Eric"` do not appear in
`eitherNeighborsOrEmployees`.
    ///
    ///     let employees: Set =
["Alicia", "Bethany", "Diana", "Eric"]
    ///     let neighbors: Set =
["Bethany", "Eric", "Forlani"]
    ///     let
```

```
eitherNeighborsOrEmployees =
employees.symmetricDifference(neighbors)
    /**
print(eitherNeighborsOrEmployees)
    /**
     // Prints "["Diana",
"Forlani", "Alicia"]"
    /**
    /**
 - Parameter other: A set of the
same type as the current set.
    /**
 - Returns: A new set.
public func symmetricDifference(_
other: AccessibilityTraits) ->
AccessibilityTraits
```

```
    /**
     // Removes the elements of the set
that are also in the given set and adds
    /**
     // the members of the given set that
are not already in the set.
    /**
    /**
     // In the following example, the
elements of the `employees` set that are
    /**
     // also members of `neighbors` are
removed from `employees`, while the
    /**
     // elements of `neighbors` that are
not members of `employees` are added to
    /**
     `employees`. In particular, the
names `'"Bethany"'` and `'"Eric"'` are
    /**
     removed from `employees` while
the name `'"Forlani"'` is added.
    /**
    /**
        var employees: Set =
["Alicia", "Bethany", "Diana", "Eric"]
    /**
        let neighbors: Set =
```

```
["Bethany", "Eric", "Forlani"]
    /**
employees.formSymmetricDifference(neighbo
rs)
    /**
        print(employees)
    /**
        // Prints "["Diana",
"Forlani", "Alicia"]"
    /**
    /**
        - Parameter other: A set of the
same type.
public mutating func
formSymmetricDifference(_ other:
AccessibilityTraits)
```

```
    /**
        Returns a Boolean value that
indicates whether the given element
exists
    /**
        in the set.
    /**
    /**
        This example uses the
`contains(_:)` method to test whether an
integer is
    /**
        a member of a set of prime
numbers.
    /**
    /**
        let primes: Set = [2, 3, 5,
7, 11, 13, 17, 19, 23, 29, 31, 37]
    /**
        let x = 5
    /**
        if primes.contains(x) {
    /**
            print("\(x) is prime!")
    /**
        } else {
    /**
            print("\(x). Not prime.")
    /**
    }
```

```
    ///      // Prints "5 is prime!"  
    ///  
    /// - Parameter member: An element to  
    look for in the set.  
    /// - Returns: `true` if `member`  
    exists in the set; otherwise, `false`.  
    public func contains(_ member:  
    AccessibilityTraits) -> Bool  
  
        /// Inserts the given element in the  
        set if it is not already present.  
        ///  
        /// If an element equal to  
        `newMember` is already contained in the  
        set, this  
        /// method has no effect. In this  
        example, a new element is inserted into  
        /// `classDays`, a set of days of the  
        week. When an existing element is  
        /// inserted, the `classDays` set  
        does not change.  
        ///  
        /// enum DayOfTheWeek: Int {  
        ///     case sunday, monday,  
        tuesday, wednesday, thursday,  
        ///         friday, saturday  
        ///     }  
        ///  
        ///     var classDays:  
Set<DayOfTheWeek> = [.wednesday, .friday]  
        ///  
print(classDays.insert(.monday))  
        ///      // Prints "(true, .monday)"
```

```
    ///      print(classDays)
    ///      // Prints
" [.friday, .wednesday, .monday]"
    ///
    ///
print(classDays.insert(.friday))
    ///      // Prints "(false, .friday)"
    ///      print(classDays)
    ///      // Prints
" [.friday, .wednesday, .monday]"
    ///
    /// - Parameter newMember: An element
to insert into the set.
    /// - Returns: `(true, newMember)` if
`newMember` was not contained in the
    /// set. If an element equal to
`newMember` was already contained in the
    /// set, the method returns
`(false, oldMember)`, where `oldMember`
is the
    /// element that was equal to
`newMember`. In some cases, `oldMember`
may
    /// be distinguishable from
`newMember` by identity comparison or
some
    /// other means.
public mutating func insert(_
newMember: AccessibilityTraits) ->
(inserted: Bool, memberAfterInsert:
AccessibilityTraits)

    /// Removes the given element and any
```

elements subsumed by the given element.

///

/// – Parameter `member`: The element of the set to remove.

/// – Returns: For ordinary sets, an element equal to `member` if `member` is

/// contained in the set; otherwise, `nil`. In some cases, a returned

/// element may be distinguishable from `member` by identity comparison

/// or some other means.

///

/// For sets where the set type and element type are the same, like

/// `OptionSet` types, this method returns any intersection between the set

/// and `[member]`, or `nil` if the intersection is empty.

```
public mutating func remove(_ member:  
AccessibilityTraits) ->  
AccessibilityTraits?
```

/// Inserts the given element into the set unconditionally.

///

/// If an element equal to `newMember` is already contained in the set,

/// `newMember` replaces the existing element. In this example, an existing

/// element is inserted into `classDays`, a set of days of the week.

```
///  
///     enum DayOfTheWeek: Int {  
///         case sunday, monday,  
tuesday, wednesday, thursday,  
///             friday, saturday  
///     }  
///  
///     var classDays:  
Set<DayOfTheWeek> = [.monday, .wednesday,  
.friday]  
///  
print(classDays.update(with: .monday))  
///     // Prints "Optional(.monday)"  
///  
/// - Parameter newMember: An element  
to insert into the set.  
/// - Returns: For ordinary sets, an  
element equal to `newMember` if the set  
/// already contained such a  
member; otherwise, `nil`. In some cases,  
the  
///     returned element may be  
distinguishable from `newMember` by  
identity  
///     comparison or some other means.  
///  
///     For sets where the set type and  
element type are the same, like  
///     `OptionSet` types, this method  
returns any intersection between the  
///     set and `[newMember]`, or `nil`  
if the intersection is empty.  
public mutating func update(with
```

```
newMember: AccessibilityTraits) ->
AccessibilityTraits?
```

```
    /// 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:
AccessibilityTraits, b:
AccessibilityTraits) -> Bool
```

```
    /// The type of the elements of an
array literal.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias ArrayLiteralElement
= AccessibilityTraits

    /// A type for which the conforming
type provides a containment test.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Element =
AccessibilityTraits
}
```

/// An alignment in both axes.

///

/// An `Alignment` contains a  
``HorizontalAlignment`` guide and a  
/// ``VerticalAlignment`` guide. Specify  
an alignment to direct the behavior of  
/// certain layout containers and  
modifiers, like when you place views in a  
/// ``ZStack``, or layer a view in front  
of or behind another view using  
/// ``View/overlay(alignment:content:)``  
or  
///

``View/background(alignment:content:)``,  
respectively. During layout,  
/// SwiftUI brings the specified guides  
of the affected views together,  
/// aligning the views.

///

/// SwiftUI provides a set of built-in  
alignments that represent common  
/// combinations of the built-in  
horizontal and vertical alignment guides.  
/// The blue boxes in the following  
diagram demonstrate the alignment named  
/// by each box's label, relative to the  
background view:

///

/// ! [A square that's divided into four  
equal quadrants. The upper-  
/// left quadrant contains the text, Some  
text in an upper quadrant. The

```
/// lower-right quadrant contains the
text, More text in a lower quadrant.
/// In both cases, the text is split over
two lines. A variety of blue
/// boxes are overlaid atop the square.
Each contains the name of a built-in
/// alignment, and is aligned with the
square in a way that matches the
/// alignment name. For example, the box
labeled center appears at the
/// center of the square.] (Alignment-1-
iOS)
///
/// The following code generates the
diagram above, where each blue box
appears
/// in an overlay that's configured with
a different alignment:
///
///     struct AlignmentGallery: View {
///         var body: some View {
///             BackgroundView()
///                 .overlay(alignment: .
topLeading) { box(".topLeading") }
///                 .overlay(alignment: .
top) { box(".top") }
///                 .overlay(alignment: .
topTrailing) { box(".topTrailing") }
///                 .overlay(alignment: .
leading) { box(".leading") }
///                 .overlay(alignment: .
center) { box(".center") }
///                 .overlay(alignment: .
```

```
trailing) { box(".trailing") }
///                               .overlay(alignment: .
bottomLeading) { box(".bottomLeading") }
///                               .overlay(alignment: .
bottom) { box(".bottom") }
///                               .overlay(alignment: .
bottomTrailing)
{ box(".bottomTrailing") }
///                               .overlay(alignment: .
leadingLastTextBaseline)
{ box(".leadingLastTextBaseline") }
///                               .overlay(alignment: .
trailingFirstTextBaseline)
{ box(".trailingFirstTextBaseline") }
///                               }

///
///           private func box(_ name:
String) -> some View {
///               Text(name)
///               .font(.system(.caption
n, design: .monospaced))
///               .padding(2)
///               .foregroundColor(.whi
te)
///               .background(.blue.o
acity(0.8), in: Rectangle())
///               }
///               }

///
///           private struct BackgroundView:
View {
///               var body: some View {
///                   Grid(horizontalSpac
ing:
```

```
0, verticalSpacing: 0) {
    ///                     GridRow {
    ///                     Text("Some text
    in an upper quadrant")
    ///                     Color.gray.opacity(0.3)
    ///                     }
    ///                     GridRow {
    ///                     Color.gray.opacity(0.3)
    ///                     Text("More text
    in a lower quadrant")
    ///                     }
    ///                     }
    ///                     .aspectRatio(1,
    contentMode: .fit)
    ///                     .foregroundColor(.seconda
    ry)
    ///                     .border(.gray)
    ///                     }
    ///                     }

/// To avoid crowding, the alignment
diagram shows only two of the available
/// text baseline alignments. The others
align as their names imply. Notice that
/// the first text baseline alignment
aligns with the top-most line of text in
/// the background view, while the last
text baseline aligns with the
/// bottom-most line. For more
information about text baseline
alignment, see
```

```
/// ``VerticalAlignment``.  
///  
/// In a left-to-right language like English, the leading and trailing  
/// alignments appear on the left and right edges, respectively. SwiftUI  
/// reverses these in right-to-left language environments. For more  
/// information, see ``HorizontalAlignment``.  
///  
/// Custom alignment  
///  
/// You can create custom alignments --- which you typically do to make use  
/// of custom horizontal or vertical guides --- by using the  
/// ``init(horizontal:vertical:)`` initializer. For example, you can combine  
/// a custom vertical guide called `firstThird` with the built-in horizontal  
/// ``HorizontalAlignment/center`` guide, and use it to configure a ``ZStack``:  
///  
///     ZStack(alignment:  
Alignment(horizontal: .center,  
vertical: .firstThird)) {  
///         // ...  
///     }  
///  
/// For more information about creating custom guides, including the code  
/// that creates the custom `firstThird`
```

```
alignment in the example above,  
/// see ``AlignmentID``.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct Alignment :  
Equatable {  
  
    /// The alignment on the horizontal  
axis.  
    ///  
    /// Set this value when you  
initialize an alignment using the  
    /// ``init(horizontal:vertical:)``  
method. Use one of the built-in  
    /// ``HorizontalAlignment`` guides,  
like ``HorizontalAlignment/center``,  
    /// or a custom guide that you  
create.  
    ///  
    /// For information about creating  
custom guides, see ``AlignmentID``.  
    public var horizontal:  
HorizontalAlignment  
  
    /// The alignment on the vertical  
axis.  
    ///  
    /// Set this value when you  
initialize an alignment using the  
    /// ``init(horizontal:vertical:)``  
method. Use one of the built-in  
    /// ``VerticalAlignment`` guides,  
like ``VerticalAlignment/center``,
```

```
    /// or a custom guide that you
create.
    /**
     /// For information about creating
custom guides, see ``AlignmentID``.
    public var vertical:
VerticalAlignment

    /// Creates a custom alignment value
with the specified horizontal
    /// and vertical alignment guides.
    /**
     /// SwiftUI provides a variety of
built-in alignments that combine built-in
    /// ``HorizontalAlignment`` and
``VerticalAlignment`` guides. Use this
    /// initializer to create a custom
alignment that makes use
    /// of a custom horizontal or
vertical guide, or both.
    /**
     /// For example, you can combine a
custom vertical guide called
    /// `firstThird` with the built-in
``HorizontalAlignment/center``
    /// guide, and use it to configure a
``ZStack``:
    /**
     ///      ZStack(alignment:
Alignment(horizontal: .center,
vertical: .firstThird)) {
    ///          // ...
    ///      }
```

```
///  
/// For more information about  
creating custom guides, including the  
code  
    /// that creates the custom  
`firstThird` alignment in the example  
above,  
    /// see ``AlignmentID``.  
///  
/// - Parameters:  
///   - horizontal: The alignment on  
the horizontal axis.  
///   - vertical: The alignment on  
the vertical axis.  
@inlinable public init(horizontal:  
HorizontalAlignment, vertical:  
VerticalAlignment)  
  
    /// 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: Alignment,  
b: Alignment) -> Bool  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Alignment {  
  
    /// A guide that marks the center of  
    the view.  
    ///  
    /// This alignment combines the  
    ``HorizontalAlignment/center``  
    /// horizontal guide and the  
    ``VerticalAlignment/center``  
    /// vertical guide:  
    ///  
    /// ! [A square that's divided into  
    four equal quadrants. The upper-  
    /// left quadrant contains the text,  
    Some text in an upper quadrant. The  
    /// lower-right quadrant contains the  
    text, More text in a lower quadrant.  
    /// In both cases, the text is split  
    over two lines. A blue box that  
    /// contains the text, Center,  
    appears at the center of the  
    /// square.] (Alignment-center-1-iOS)  
    public static let center: Alignment  
  
    /// A guide that marks the leading  
    edge of the view.  
    ///  
    /// This alignment combines the  
    ``HorizontalAlignment/leading``  
    /// horizontal guide and the
```

```
``VerticalAlignment/center``
    /// vertical guide:
    ///
    /// ![A square that's divided into
four equal quadrants. The upper-
    /// left quadrant contains the text,
Some text in an upper quadrant. The
    /// lower-right quadrant contains the
text, More text in a lower quadrant.
    /// In both cases, the text is split
over two lines. A blue box that
    /// contains the text, Leading,
appears on the left edge of the
    /// square, centered vertically.]
```

(Alignment-leading-1-iOS)

```
public static let leading: Alignment

    /// A guide that marks the trailing
edge of the view.
    ///
    /// This alignment combines the
``HorizontalAlignment/trailing``
    /// horizontal guide and the
``VerticalAlignment/center``
    /// vertical guide:
    ///
    /// ![A square that's divided into
four equal quadrants. The upper-
    /// left quadrant contains the text,
Some text in an upper quadrant. The
    /// lower-right quadrant contains the
text, More text in a lower quadrant.
    /// In both cases, the text is split
```

```
over two lines. A blue box that
    /// contains the text, Trailing,
appears on the right edge of the
    /// square, centered vertically.]
```

(Alignment-trailing-1-iOS)

```
public static let trailing: Alignment
```

  

```
    /// A guide that marks the top edge
of the view.
```

```
    ///
```

```
    /// This alignment combines the
``HorizontalAlignment/center``
    /// horizontal guide and the
``VerticalAlignment/top``
    /// vertical guide:
```

```
    ///
```

```
    /// ![A square that's divided into
four equal quadrants. The upper-
    /// left quadrant contains the text,
Some text in an upper quadrant. The
    /// lower-right quadrant contains the
text, More text in a lower quadrant.
    /// In both cases, the text is split
over two lines. A blue box that
    /// contains the text, Top, appears
on the top edge of the
    /// square, centered horizontally.]
```

(Alignment-top-1-iOS)

```
public static let top: Alignment
```

  

```
    /// A guide that marks the bottom
edge of the view.
```

```
    ///
```

```
    /// This alignment combines the
``HorizontalAlignment/center``
    /// horizontal guide and the
``VerticalAlignment/bottom``
    /// vertical guide:
    ///
    /// ![A square that's divided into
four equal quadrants. The upper-
    /// left quadrant contains the text,
Some text in an upper quadrant. The
    /// lower-right quadrant contains the
text, More text in a lower quadrant.
    /// In both cases, the text is split
over two lines. A blue box that
    /// contains the text, Bottom,
appears on the bottom edge of the
    /// square, centered horizontally.]
```

(Alignment-bottom-1-iOS)

**public static let bottom: Alignment**

```
    /// A guide that marks the top and
leading edges of the view.
    ///
    /// This alignment combines the
``HorizontalAlignment/leading``
    /// horizontal guide and the
``VerticalAlignment/top``
    /// vertical guide:
    ///
    /// ![A square that's divided into
four equal quadrants. The upper-
    /// left quadrant contains the text,
Some text in an upper quadrant. The
```

```
    /// lower-right quadrant contains the
text, More text in a lower quadrant.
    /// In both cases, the text is split
over two lines. A blue box that
    /// contains the text, topLeading,
appears in the upper-left corner of
    /// the square.](Alignment-
topLeading-1-iOS)
public static let topLeading:
Alignment
```

```
    /// A guide that marks the top and
trailing edges of the view.
    ///
    /// This alignment combines the
``HorizontalAlignment/trailing``
    /// horizontal guide and the
``VerticalAlignment/top``
    /// vertical guide:
    ///
    /// ![A square that's divided into
four equal quadrants. The upper-
    /// left quadrant contains the text,
Some text in an upper quadrant. The
    /// lower-right quadrant contains the
text, More text in a lower quadrant.
    /// In both cases, the text is split
over two lines. A blue box that
    /// contains the text, topTrailing,
appears in the upper-right corner of
    /// the square.](Alignment-
topTrailing-1-iOS)
public static let topTrailing:
```

## Alignment

```
    /// A guide that marks the bottom and
    leading edges of the view.
    ///
    /// This alignment combines the
``HorizontalAlignment/leading``
    /// horizontal guide and the
``VerticalAlignment/bottom``
    /// vertical guide:
    ///
    /// ! [A square that's divided into
four equal quadrants. The upper-
    /// left quadrant contains the text,
Some text in an upper quadrant. The
    /// lower-right quadrant contains the
text, More text in a lower quadrant.
    /// In both cases, the text is split
over two lines. A blue box that
    /// contains the text, bottomLeading,
appears in the lower-left corner of
    /// the square.] (Alignment-
bottomLeading-1-iOS)
public static let bottomLeading:
Alignment
```

```
    /// A guide that marks the bottom and
trailing edges of the view.
    ///
    /// This alignment combines the
``HorizontalAlignment/trailing``
    /// horizontal guide and the
``VerticalAlignment/bottom``
```

```
    /// vertical guide:  
    ///  
    /// ! [A square that's divided into  
    four equal quadrants. The upper-  
    /// left quadrant contains the text,  
    Some text in an upper quadrant. The  
    /// lower-right quadrant contains the  
    text, More text in a lower quadrant.  
    /// In both cases, the text is split  
    over two lines. A blue box that  
    /// contains the text,  
    bottomTrailing, appears in the lower-  
    right corner of  
    /// the square.] (Alignment-  
bottomTrailing-1-iOS)  
    public static let bottomTrailing:  
        Alignment  
    }  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Alignment {  
  
    /// A guide that marks the top-most  
    text baseline in a view.  
    ///  
    /// This alignment combines the  
    ``HorizontalAlignment/center``  
    /// horizontal guide and the  
    ``VerticalAlignment/firstTextBaseline``  
    /// vertical guide:  
    ///  
    /// ! [A square that's divided into
```

four equal quadrants. The upper-  
    /// left quadrant contains the text,  
Some text in an upper quadrant. The  
    /// lower-right quadrant contains the  
text, More text in a lower quadrant.  
    /// In both cases, the text is split  
over two lines. A blue box that  
    /// contains the text,  
centerFirstTextBaseline, appears aligned  
with, and  
    /// partially overlapping, the first  
line of the text in the upper quadrant,  
    /// centered horizontally.]

(Alignment–centerFirstTextBaseline–1–iOS)

```
public static var  
centerFirstTextBaseline: Alignment {  
get }
```

    /// A guide that marks the bottom-  
most text baseline in a view.  
    ///  
    /// This alignment combines the  
``HorizontalAlignment/center``  
    /// horizontal guide and the  
``VerticalAlignment/lastTextBaseline``  
    /// vertical guide:  
    ///  
    /// ![A square that's divided into  
four equal quadrants. The upper-  
    /// left quadrant contains the text,  
Some text in an upper quadrant. The  
    /// lower-right quadrant contains the  
text, More text in a lower quadrant.

```
    /// In both cases, the text is split  
    over two lines. A blue box that  
        /// contains the text,  
centerLastTextBaseline, appears aligned  
with, and  
        /// partially overlapping, the last  
line of the text in the lower quadrant,  
        /// centered horizontally.]  
(Alignment-centerLastTextBaseline-1-iOS)  
    public static var  
centerLastTextBaseline: Alignment { get }
```

```
    /// A guide that marks the leading  
edge and top-most text baseline in a  
        /// view.  
    ///  
        /// This alignment combines the  
``HorizontalAlignment/leading``  
        /// horizontal guide and the  
``VerticalAlignment/firstTextBaseline``  
        /// vertical guide:  
    ///  
        /// ![A square that's divided into  
four equal quadrants. The upper-  
        /// left quadrant contains the text,  
Some text in an upper quadrant. The  
        /// lower-right quadrant contains the  
text, More text in a lower quadrant.  
        /// In both cases, the text is split  
over two lines. A blue box that  
        /// contains the text,  
leadingFirstTextBaseline, appears aligned  
with, and
```

```
    /// partially overlapping, the first
    line of the text in the upper quadrant.
    /// The box aligns with the left edge
    of the
    /// square.])(Alignment-
    leadingFirstTextBaseline-1-iOS)
public static var
leadingFirstTextBaseline: Alignment { get
}

    /// A guide that marks the leading
    edge and bottom-most text baseline
    /// in a view.
    ///
    /// This alignment combines the
    ``HorizontalAlignment/leading``
    /// horizontal guide and the
    ``VerticalAlignment/lastTextBaseline``
    /// vertical guide:
    ///
    /// ![A square that's divided into
    four equal quadrants. The upper-
    /// left quadrant contains the text,
    Some text in an upper quadrant. The
    /// lower-right quadrant contains the
    text, More text in a lower quadrant.
    /// In both cases, the text is split
    over two lines. A blue box that
    /// contains the text,
    leadingLastTextBaseline, appears aligned
    with the
    /// last line of the text in the
    lower quadrant. The box aligns with the
```

```
    /// left edge of the square.]  
(Alignment-leadingLastTextBaseline-1-iOS)  
    public static var  
leadingLastTextBaseline: Alignment {  
get }
```

```
    /// A guide that marks the trailing  
edge and top-most text baseline in  
    /// a view.  
    ///  
    /// This alignment combines the  
``HorizontalAlignment/trailing``  
    /// horizontal guide and the  
``VerticalAlignment/firstTextBaseline``  
    /// vertical guide:  
    ///  
    /// ![A square that's divided into  
four equal quadrants. The upper-  
    /// left quadrant contains the text,  
Some text in an upper quadrant. The  
    /// lower-right quadrant contains the  
text, More text in a lower quadrant.  
    /// In both cases, the text is split  
over two lines. A blue box that  
    /// contains the text,  
trailingFirstTextBaseline, appears  
aligned with the  
    /// first line of the text in the  
upper quadrant. The box aligns with the  
    /// right edge of the square.]  
(Alignment-trailingFirstTextBaseline-1-  
iOS)  
    public static var
```

```
trailingFirstTextBaseline: Alignment {
get }

    /// A guide that marks the trailing
    edge and bottom-most text baseline
    /// in a view.
    ///
    /// This alignment combines the
``HorizontalAlignment/trailing``
    /// horizontal guide and the
``VerticalAlignment/lastTextBaseline``
    /// vertical guide:
    ///
    /// ! [A square that's divided into
four equal quadrants. The upper-
    /// left quadrant contains the text,
Some text in an upper quadrant. The
    /// lower-right quadrant contains the
text, More text in a lower quadrant.
    /// In both cases, the text is split
over two lines. A blue box that
    /// contains the text,
trailingLastTextBaseline, appears aligned
with the
    /// last line of the text in the
lower quadrant. The box aligns with the
    /// right edge of the square.]  

(Alignment-trailingLastTextBaseline-1-
iOS)
    public static var
trailingLastTextBaseline: Alignment { get
}
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Alignment : Sendable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Alignment : BitwiseCopyable {  
}  
  
/// A type that you use to create custom  
alignment guides.  
///  
/// Every built-in alignment guide that  
``VerticalAlignment`` or  
/// ``HorizontalAlignment`` defines as a  
static property, like  
/// ``VerticalAlignment/top`` or  
``HorizontalAlignment/leading``, has a  
/// unique alignment identifier type that  
produces the default offset for  
/// that guide. To create a custom  
alignment guide, define your own  
alignment  
/// identifier as a type that conforms to  
the `AlignmentID` protocol, and  
/// implement the required  
``AlignmentID/defaultValue(in:)`` method:  
///  
///     private struct  
FirstThirdAlignment: AlignmentID {  
///         static func defaultValue(in
```

```
context: ViewDimensions) -> CGFloat {
    ///             context.height / 3
    ///         }
    ///     }
    ///
    /// When implementing the method,
    calculate the guide's default offset
    /// from the view's origin. If it's
    helpful, you can use information from the
    /// ``ViewDimensions`` input in the
    calculation. This parameter provides
    context
    /// about the specific view that's using
    the guide. The above example creates an
    /// identifier called
    `FirstThirdAlignment` and calculates a
    default value
    /// that's one-third of the height of the
    aligned view.
    ///
    /// Use the identifier's type to create a
    static property in an extension of
    /// one of the alignment guide types,
    like ``VerticalAlignment``:
    ///
    ///     extension VerticalAlignment {
    ///         static let firstThird =
    VerticalAlignment(FirstThirdAlignment.sel
    f)
    ///     }
    ///
    /// You can apply your custom guide like
    any of the built-in guides. For
```

```
/// example, you can use an ``HStack`` to
/// align its views at one-third
/// of their height using the guide
defined above:
///
///     struct StripesGroup: View {
///         var body: some View {
///
HStack(alignment: .firstThird, spacing:
1) {
///
HorizontalStripes().frame(height: 60)
///
HorizontalStripes().frame(height: 120)
///
HorizontalStripes().frame(height: 90)
///
}
///
}
///

///     struct HorizontalStripes: View {
///         var body: some View {
///             VStack(spacing: 1) {
///                 ForEach(0..<3) { _ in
Color.blue }
///
}
///
}
///

/// Because each set of stripes has three
equal, vertically stacked
/// rectangles, they align at the bottom
edge of the top rectangle. This
```

```
/// corresponds in each case to a third
/// of the overall height, as
/// measured from the origin at the top
/// of each set of stripes:
///
/// ! [Three vertical stacks of
/// rectangles, arranged in a row.
/// The rectangles in each stack have the
/// same height as each other, but
/// different heights than the rectangles
/// in the other stacks. The bottom edges
/// of the top-most rectangle in each
/// stack are aligned with each
/// other.] (AlignmentId-1-iOS)
///
/// You can also use the
``View/alignmentGuide(_:computeValue:)``
view
/// modifier to alter the behavior of
your custom guide for a view, as you
/// might alter a built-in guide. For
example, you can change
/// one of the stacks of stripes from the
previous example to align its
/// `firstThird` guide at two thirds of
the height instead:
///
/// struct StripesGroupModified: View
{
///     var body: some View {
///
HStack(alignment: .firstThird, spacing:
1) {
```

```
///  
HorizontalStripes().frame(height: 60)  
///  
HorizontalStripes().frame(height: 120)  
///  
HorizontalStripes().frame(height: 90)  
/// .alignmentGuide(.  
firstThird) { context in  
/// 2 *  
context.height / 3  
/// }  
/// }  
/// }  
/// }  
///  
/// The modified guide calculation causes  
the affected view to place the  
/// bottom edge of its middle rectangle  
on the `firstThird` guide, which aligns  
/// with the bottom edge of the top  
rectangle in the other two groups:  
///  
/// ! [Three vertical stacks of  
rectangles, arranged in a row.  
/// The rectangles in each stack have the  
same height as each other, but  
/// different heights than the rectangles  
in the other stacks. The bottom edges  
/// of the top-most rectangle in the  
first two stacks are aligned with each  
/// other, and with the bottom edge of  
the middle rectangle in the third  
/// stack.] (AlignmentId-2-iOS)
```

```
///  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
public protocol AlignmentID {  
  
    /// Calculates a default value for  
    the corresponding guide in the specified  
    /// context.  
    ///  
    /// Implement this method when you  
    create a type that conforms to the  
    /// ``AlignmentID`` protocol. Use the  
    method to calculate the default  
    /// offset of the corresponding  
    alignment guide. SwiftUI interprets the  
    /// value that you return as an  
    offset in the coordinate space of the  
    /// view that's being laid out. For  
    example, you can use the context to  
    /// return a value that's one-third  
    of the height of the view:  
    ///  
    ///     private struct  
FirstThirdAlignment: AlignmentID {  
    ///         static func  
defaultValue(in context: ViewDimensions)  
-> CGFloat {  
    ///             context.height / 3  
    ///         }  
    ///     }  
    ///  
    /// You can override the default  
    value that this method returns for a
```

```
    /// particular guide by adding the
    ///
``View/alignmentGuide(_:computeValue:)``
view modifier to a
    /// particular view.
    ///
    /// - Parameter context: The context
of the view that you apply
    /// the alignment guide to. The
context gives you the view's dimensions,
    /// as well as the values of other
alignment guides that apply to the
    /// view, including both built-in
and custom guides. You can use any of
    /// these values, if helpful, to
calculate the value for your custom
    /// guide.
    ///
    /// - Returns: The offset of the
guide from the origin in the
    /// view's coordinate space.
static func defaultValue(in context:
ViewDimensions) -> CGFloat
}
```

```
/// A single sort key type for alignment
guides in both axes.
///
/// You don't use this type directly.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct AlignmentKey :  
Hashable, Comparable {
```

```
    /// Returns a Boolean value  
indicating whether the value of the first  
    /// argument is less than that of the  
second argument.
```

```
    ///  
    /// This function is the only  
requirement of the `Comparable` protocol.  
The
```

```
    /// remainder of the relational  
operator functions are implemented by the  
    /// standard library for any type  
that conforms to `Comparable`.
```

```
    ///  
    /// - Parameters:  
    ///   - lhs: A value to compare.  
    ///   - rhs: Another value to  
compare.
```

```
    public static func < (lhs:  
AlignmentKey, rhs: AlignmentKey) -> Bool
```

```
    /// 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:
AlignmentKey, b: AlignmentKey) -> 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 }
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension AlignmentKey : Sendable {  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension AlignmentKey : BitwiseCopyable  
{  
}
```

    /// An opaque value derived from an  
anchor source and a particular view.  
    ///  
    /// You can convert the anchor to a  
`Value` in the coordinate space of a  
target  
    /// view by using a ``GeometryProxy`` to

```
specify the target view.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct Anchor<Value> {  
  
    /// A type-erased geometry value that  
    produces an anchored value of a given  
    /// type.  
    ///  
    /// SwiftUI passes anchored geometry  
    values around the view tree via  
    /// preference keys. It then converts  
    them back into the local coordinate  
    /// space using a ``GeometryProxy``  
    value.  
    @frozen public struct Source {  
    }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Anchor : Sendable where Value :  
Sendable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Anchor : Equatable where  
Value : Equatable {  
  
    /// 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 == (lhs:  
Anchor<Value>, rhs: Anchor<Value>) ->  
Bool  
}
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Anchor : Hashable where Value :  
Hashable {
```

```
    /// 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)
```

```
    /// 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 }
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
```

```
extension Anchor.Source : Sendable where
    Value : Sendable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Anchor.Source {

    public init<T>(_ array:
        [Anchor<T>.Source]) where Value == [T]
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Anchor.Source {

    public init<T>(_ anchor:
        Anchor<T>.Source?) where Value == T?
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Anchor.Source where Value ==
CGPoint {

    public static func point(_ p:
        CGPoint) -> Anchor<Value>.Source
}

    public static func unitPoint(_ p:
        UnitPoint) -> Anchor<Value>.Source

    public static var topLeading:
        Anchor<CGPoint>.Source { get }
```

```
    public static var top:  
Anchor<CGPoint>.Source { get }  
  
    public static var topTrailing:  
Anchor<CGPoint>.Source { get }  
  
    public static var leading:  
Anchor<CGPoint>.Source { get }  
  
    public static var center:  
Anchor<CGPoint>.Source { get }  
  
    public static var trailing:  
Anchor<CGPoint>.Source { get }  
  
    public static var bottomLeading:  
Anchor<CGPoint>.Source { get }  
  
    public static var bottom:  
Anchor<CGPoint>.Source { get }  
  
    public static var bottomTrailing:  
Anchor<CGPoint>.Source { get }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Anchor.Source where Value ==  
CGRect {  
  
    /// Returns an anchor source rect  
    /// defined by `r` in the current view.  
}
```

```
    public static func rect(_ r: CGRect)
-> Anchor<Value>.Source

        /// An anchor source rect defined as
the entire bounding rect of the current
        /// view.
        public static var bounds:
Anchor<CGRect>.Source { get }
}

/// A geometric angle whose value you
access in either radians or degrees.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct Angle {

    public var radians: Double

    @inlinable public var degrees: Double

    @inlinable public init()

    @inlinable public init(radians:
Double)

    @inlinable public init(degrees:
Double)

    @inlinable public static func
radians(_ radians: Double) -> Angle

    @inlinable public static func
degrees(_ degrees: Double) -> Angle
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Angle : Hashable, Comparable {  
  
    /// Returns a Boolean value  
    /// indicating whether the value of the first  
    /// argument is less than that of the  
    /// second argument.  
    ///  
    /// This function is the only  
    requirement of the `Comparable` protocol.  
    The  
    /// remainder of the relational  
    operator functions are implemented by the  
    /// standard library for any type  
    that conforms to `Comparable`.  
    ///  
    /// - Parameters:  
    /// - lhs: A value to compare.  
    /// - rhs: Another value to  
    compare.  
    @inlinable public static func < (lhs:  
        Angle, rhs: Angle) -> Bool  
  
    /// 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: Angle, b:
```

```
Angle) -> 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 }  
    }  
  
    @available(iOS 13.0, macOS 10.15, tvOS  
    13.0, watchOS 6.0, *)  
    extension Angle : Animatable {  
  
        /// The data to animate.  
        public var animatableData: Double  
  
        @inlinable public static var zero:  
        Angle { get }  
  
        /// The type defining the data to  
        animate.  
        @available(iOS 13.0, tvOS 13.0,
```

```
watchOS 6.0, macOS 10.15, *)
    public typealias AnimatableData =
Double
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Angle : Sendable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Angle : BitwiseCopyable {
}

/// An angular gradient.
///
/// An angular gradient is also known as
/// a "conic" gradient. This gradient
/// applies the color function as the
/// angle changes, relative to a center
/// point and defined start and end
/// angles. If `endAngle - startAngle > 2π` ,
/// the gradient only draws the last
/// complete turn. If
/// `endAngle - startAngle < 2π` , the
/// gradient fills the missing area with
/// the colors defined by gradient
/// locations one and zero, transitioning
/// between the two halfway across the
/// missing area. The gradient maps the
/// unit space center point into the
/// bounding rectangle of each shape filled
```

```
    /// with the gradient.  
    ///  
    /// When using an angular gradient as a  
    shape style, you can also use  
    ///  
    ``ShapeStyle/angularGradient(_:center:sta  
rtAngle:endAngle:)``,  
    ///  
    ``ShapeStyle/conicGradient(_:center:angle  
:)``, or similar methods.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct AngularGradient :  
ShapeStyle, View, Sendable {  
  
    /// Creates an angular gradient.  
    public init(gradient: Gradient,  
    center: UnitPoint, startAngle: Angle  
= .zero, endAngle: Angle = .zero)  
  
        /// Creates an angular gradient from  
        a collection of colors.  
        public init(colors: [Color], center:  
UnitPoint, startAngle: Angle, endAngle:  
Angle)  
  
        /// Creates an angular gradient from  
        a collection of color stops.  
        public init(stops: [Gradient.Stop],  
center: UnitPoint, startAngle: Angle,  
endAngle: Angle)  
  
    /// Creates a conic gradient that
```

completes a full turn.

```
public init(gradient: Gradient,  
center: UnitPoint, angle: Angle = .zero)
```

```
    /// Creates a conic gradient from a  
collection of colors that completes  
    /// a full turn.
```

```
public init(colors: [Color], center:  
UnitPoint, angle: Angle = .zero)
```

```
    /// Creates a conic gradient from a  
collection of color stops that
```

```
    /// completes a full turn.
```

```
public init(stops: [Gradient.Stop],  
center: UnitPoint, angle: Angle = .zero)
```

```
    /// The type of view representing the  
body of this view.
```

```
///
```

```
    /// When you create a custom view,  
Swift infers this type from your
```

```
    /// implementation of the required  
``View/body-swift.property`` property.
```

```
    @available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)
```

```
    public typealias Body
```

```
    /// The type of shape style this will  
resolve to.
```

```
///
```

```
    /// When you create a custom shape  
style, Swift infers this type
```

```
    /// from your implementation of the
```

```
required `resolve` function.
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Resolved = Never
}

/// A type that describes how to animate
a property of a view.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public protocol Animatable {

    /// The type defining the data to
animate.
    associatedtype AnimatableData : VectorArithmetic

    /// The data to animate.
    var animatableData:
Self.AnimatableData { get set }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Animatable where Self :
VectorArithmetic {

    /// The data to animate.
    public var animatableData: Self
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
```

```
extension Animatable where
Self.AnimatableData ==
EmptyAnimatableData {

    /// The data to animate.
    public var animatableData:
EmptyAnimatableData
}

/// A pair of animatable values, which is
itself animatable.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct
AnimatablePair<First, Second> :
VectorArithmetic where First :
VectorArithmetic, Second :
VectorArithmetic, Second :
VectorArithmetic {

    /// The first value.
    public var first: First

    /// The second value.
    public var second: Second

    /// Creates an animated pair with the
provided values.
    @inlinable public init(_ first:
First, _ second: Second)

    /// The zero value.
    ///
    /// Zero is the identity element for
```

```
addition. For any value,
    /// `x + .zero == x` and `.zero + x
== x`.
    public static var zero:
AnimatablePair<First, Second> { get }

    /// Adds two values and stores the
result in the left-hand-side variable.
    ///
    /// - Parameters:
    ///   - lhs: The first value to add.
    ///   - rhs: The second value to add.
    public static func += (lhs: inout
AnimatablePair<First, Second>, rhs:
AnimatablePair<First, Second>)

    /// Subtracts the second value from
the first and stores the difference in
the
    /// left-hand-side variable.
    ///
    /// - Parameters:
    ///   - lhs: A numeric value.
    ///   - rhs: The value to subtract
from `lhs`.
    public static func -= (lhs: inout
AnimatablePair<First, Second>, rhs:
AnimatablePair<First, Second>)

    /// Adds two values and produces
their sum.
    ///
    /// The addition operator (`+`)
```

calculates the sum of its two arguments.

For

```
    /// example:  
    ///  
    ///      1 + 2          // 3  
    ///      -10 + 15       // 5  
    ///      -15 + -5        //  
-20  
    ///      21.5 + 3.25     //  
24.75  
    ///  
    /// You cannot use `+` with arguments  
of different types. To add values of  
    /// different types, convert one of  
the values to the other value's type.
```

```
    ///  
    ///      let x: Int8 = 21  
    ///      let y: Int = 1000000  
    ///      Int(x) + y      //  
1000021  
    ///  
    /// - Parameters:  
    ///   - lhs: The first value to add.  
    ///   - rhs: The second value to add.  
public static func + (lhs:  
AnimatablePair<First, Second>, rhs:  
AnimatablePair<First, Second>) ->  
AnimatablePair<First, Second>
```

/// Subtracts one value from another  
and produces their difference.

```
    ///  
    /// The subtraction operator (`-`)
```

calculates the difference of its two  
    /// arguments. For example:

    ///  
    ///     8 - 3                                  // 5  
    ///     -10 - 5                                //  
-15  
105  
-89.5  
    ///  
    /// You cannot use ` - ` with arguments  
of different types. To subtract values  
    /// of different types, convert one  
of the values to the other value's type.

    ///  
    ///     let x: UInt8 = 21  
    ///     let y: UInt = 1000000  
    ///     y - UInt(x)                            //  
999979  
    ///  
    /// - Parameters:  
    ///     - lhs: A numeric value.  
    ///     - rhs: The value to subtract  
from `lhs`.

    public static func - (lhs:  
AnimatablePair<First, Second>, rhs:  
AnimatablePair<First, Second>) ->  
AnimatablePair<First, Second>

    /// Multiplies each component of this  
value by the given value.

    public mutating func scale(by rhs:

```
Double)
```

```
    /// The dot-product of this animated
    pair with itself.
    public var magnitudeSquared: Double {
get }

    /// 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:
AnimatablePair<First, Second>, b:
AnimatablePair<First, Second>) -> Bool
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension AnimatablePair : Sendable where
First : Sendable, Second : Sendable {

}

/// The way a view changes over time to
create a smooth visual transition from
/// one state to another.
```

```
///  
/// An `Animation` provides a visual  
transition of a view when a state value  
/// changes from one value to another.  
The characteristics of this transition  
/// vary according to the animation type.  
For instance, a ``linear`` animation  
/// provides a mechanical feel to the  
animation because its speed is consistent  
/// from start to finish. In contrast, an  
animation that uses easing, like  
/// ``easeOut``, offers a more natural  
feel by varying the acceleration  
/// of the animation.  
///  
/// To apply an animation to a view, add  
the ``View/animation(_:_value:)``  
/// modifier, and specify both an  
animation type and the value to animate.  
For  
/// instance, the ``Circle`` view in the  
following code performs an  
/// ``easeIn`` animation each time the  
state variable `scale` changes:  
///  
///     struct ContentView: View {  
///         @State private var scale =  
0.5  
///  
///         var body: some View {  
///             VStack {  
///                 Circle()  
///                     .scaleEffect(scal
```

```
e)
/// .animation(.easeIn
n, value: scale)
///
/// HStack {
///
///     Button("+")
///
///     Button("-")
///
{ scale += 0.1 }
///
{ scale -= 0.1 }
///
}
///
}
///
.padding()
///
}

///
/// @Video(source: "animation-01-
/// overview-easein.mp4", poster:
/// "animation-01-overview-easein.png", alt:
/// "A video that shows a circle enlarging
/// then shrinking to its original size using
/// an ease-in animation.")
///
/// When the value of `scale` changes,
/// the modifier
/// ``View/scaleEffect(_:anchor:)``
/// resizes ``Circle`` according to the
/// new value. SwiftUI can animate the
/// transition between sizes because
/// ``Circle`` conforms to the ``Shape``
/// protocol. Shapes in SwiftUI conform to
/// the ``Animatable`` protocol, which
/// describes how to animate a property of a
/// view.
///
/// In addition to adding an animation to
```

a view, you can also configure the  
/// animation by applying animation  
modifiers to the animation type. For  
/// example, you can:

///

/// - Delay the start of the animation by  
using the ``delay(\_:)`` modifier.

/// - Repeat the animation by using the  
``repeatCount(\_:autoreverses:)`` or  
/// ``repeatForever(autoreverses:)``  
modifiers.

/// - Change the speed of the animation  
by using the ``speed(\_:)`` modifier.

///

/// For example, the ``Circle`` view in  
the following code repeats  
/// the ``easeIn`` animation three times  
by using the  
/// ``repeatCount(\_:autoreverses:)``  
modifier:

///

```
/// struct ContentView: View {  
///     @State private var scale =  
0.5  
///  
///     var body: some View {  
///         VStack {  
///             Circle()  
///                 .scaleEffect(scal  
e)  
///                 .animation(.easeI  
n.repeatCount(3), value: scale)  
///         HStack {
```

```
///                         Button("+")
{ scale += 0.1 }
///                         Button("-")
{ scale -= 0.1 }
///                         }
///                         }
///                         .padding()
///                         }
///                         }
///                         }

/// @Video(source: "animation-02-
/// overview-easein-repeat.mp4", poster:
/// "animation-02-overview-easein-
/// repeat.png", alt: "A video that shows a
/// circle that repeats the ease-in animation
/// three times: enlarging, then shrinking,
/// then enlarging again. The animation
/// reverses causing the circle to shrink,
/// then enlarge, then shrink to its original
/// size.")
///
/// A view can also perform an animation
/// when a binding value changes. To
/// specify the animation type on a
/// binding, call its
``Binding/animation(_:)``
/// method. For example, the view in the
/// following code performs a
/// ``linear`` animation, moving the box
/// truck between the leading and trailing
/// edges of the view. The truck moves
/// each time a person clicks the ``Toggle``
/// control, which changes the value of
```

```
the `$isTrailing` binding.

////
////     struct ContentView: View {
////         @State private var isTrailing
= false
////
////         var body: some View {
////             VStack(alignment:
isTrailing ? .trailing : .leading) {
////                 Image(systemName:
"box.truck")
////                     .font(.system(siz
e: 64))
////
////                     Toggle("Move to
trailing edge",
////                         isOn:
$isTrailing.animation(.linear))
////                     }
////                 }
////             }
////
////             @Video(source: "animation-03-
overview-binding.mp4", poster:
"animation-03-overview-binding.png", alt:
"A video that shows a box truck that
moves from the leading edge of a view to
the trailing edge. The box truck then
returns to the view's leading edge.")
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct Animation :
Equatable, Sendable {
```

```
    /// Create an `Animation` that
    contains the specified custom animation.
    @available(iOS 17.0, macOS 14.0, tvOS
    17.0, watchOS 10.0, *)
    public init<A>(_ base: A) where A :
    CustomAnimation

        /// 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 == (lhs:
    Animation, rhs: Animation) -> Bool
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Animation {

    /// A default animation instance.
    ///
    /// The `default` animation is
    ``spring(response:dampingFraction:blendDu
    ration:)``
```

```
    /// with:  
    ///  
    /// - `response` equal to `0.55`  
    /// - `dampingFraction` equal to  
`1.0`  
    /// - `blendDuration` equal to `0.0`  
    ///  
    /// Prior to iOS 17, macOS 14, tvOS  
17, and watchOS 10, the `default`  
    /// animation is ``easeInOut``.  
    ///  
    /// The global function  
    /// ``withAnimation(_:_:)`` uses the  
default animation if you don't  
    /// provide one. For instance, the  
following code listing shows  
    /// an example of using the `default`  
animation to flip the text "Hello"  
    /// each time someone clicks the  
Animate button.  
    ///  
    ///     struct ContentView: View {  
    ///         @State private var  
degrees = Double.zero  
    ///  
    ///         var body: some View {  
    ///             VStack {  
    ///                 Spacer()  
    ///                 Text("Hello")  
    ///                     .font(.largeT  
itle)  
    ///                     .rotation3DEf  
fect(.degrees(degrees), axis: (x: 0, y:
```



```
    /// button.  
    ///  
    ///     struct ContentView: View {  
    ///         @State private var  
degrees = Double.zero  
    ///  
    ///             var body: some View {  
    ///                 VStack {  
    ///                     Spacer()  
    ///                     Text("Hello")  
    ///                         .font(.largeT  
title)  
    ///                         .rotationEffe  
ct(.degrees(degrees))  
    ///                         .animation(.d  
efault, value: degrees)  
    ///  
    ///                     Spacer()  
    ///                     Button("Animate")  
{  
    ///                         degrees =  
(degrees == .zero) ? 360 : .zero  
    ///                     }  
    ///                 }  
    ///             }  
    ///         }  
    ///     }  
    /// @Video(source: "animation-05-  
default-spin.mp4", poster: "animation-05-  
default-spin.png", alt: "A video that  
shows the word Hello spinning clockwise  
for one full rotation, that is, 360  
degrees. Then Hello spins
```

```
counterclockwise for one full rotation.")  
    ///  
    /// A `default` animation instance is  
only equal to other `default`  
    /// animation instances (using `==`),  
and not equal to other animation  
    /// instances even when the  
animations are identical. For example, if  
you  
    /// create an animation using the  
``spring(response:dampingFraction:blendDu  
ration:)``  
    /// modifier with the same parameter  
values that `default` uses, the  
    /// animation isn't equal to  
`default`. This behavior lets you  
    /// differentiate between animations  
that you intentionally choose and  
    /// those that use the `default`  
animation.  
    public static let `default`:  
Animation  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Animation {  
  
    /// Changes the duration of an  
animation by adjusting its speed.  
    ///  
    /// Setting the speed of an animation  
changes the duration of the animation
```

```
    /// by a factor of `speed`. A higher
    speed value causes a faster animation
    /// sequence due to a shorter
duration. For example, a one-second
animation
    /// with a speed of `2.0` completes
in half the time (half a second).
    ///
    /// struct ContentView: View {
    ///     @State private var
adjustBy = 100.0
    ///
    ///     private var
oneSecondAnimation: Animation {
    ///         .easeInOut(duration:
1.0)
    ///
    ///
    ///         var body: some View {
    ///             VStack(spacing: 40) {
    ///                 HStack(alignment:
.bottom) {
        ///
        ///             Capsule()
        ///
        ///             .frame(wi
dth: 50, height: 175 - adjustBy)
        ///
        ///             Capsule()
        ///
        ///             .frame(wi
dth: 50, height: 175 + adjustBy)
        ///
        ///         }
        ///
        ///         .animation(oneSec
ondAnimation.speed(2.0), value: adjustBy)
    ///
    ///         Button("Animate")
```

```
{  
    /// adjustBy *=  
-1  
    /// }  
    /// }  
    /// }  
    /// }  
    /// @Video(source: "animation-18-  
speed.mp4", poster: "animation-18-  
speed.png", alt: "A video that shows two  
capsules side by side that animate using  
the ease-in ease-out animation. The  
capsule on the left is short, while the  
capsule on the right is tall. They  
animate for half a second with the short  
capsule growing upwards to match the  
height of the tall capsule. Then the tall  
capsule shrinks to match the original  
height of the short capsule. For another  
half second, the capsule on the left  
shrinks to its original height, followed  
by the capsule on the right growing to  
its original height.")  
    ///  
    /// Setting `speed` to a lower number  
slows the animation, extending its  
    /// duration. For example, a one-  
second animation with a speed of `0.25`  
    /// takes four seconds to complete.  
    ///  
    /// struct ContentView: View {  
    ///     @State private var
```

```
adjustBy = 100.0
    /**
     */// private var
oneSecondAnimation: Animation {
    /**
     */// .easeInOut(duration:
1.0)
    /**
     */
    /**
     */// var body: some View {
    /**
     */// VStack(spacing: 40) {
    /**
     */// HStack(alignment:
.bottom) {
    /**
     */// Capsule()
    /**
     */// .frame(wi
dth: 50, height: 175 - adjustBy)
    /**
     */// Capsule()
    /**
     */// .frame(wi
dth: 50, height: 175 + adjustBy)
    /**
     */// }
    /**
     */// .animation(oneSec
ondAnimation.speed(0.25), value:
adjustBy)
    /**
     */
    /**
     */// Button("Animate")
{
    /**
     */// adjustBy *=
-1
    /**
     */
    /**
     */// }
    /**
     */// }
    /**
     */// }
    /**
     */
    /**
     */// @Video(source: "animation-19-
```

speed-slow.mp4", poster: "animation-19-speed-slow.png", alt: "A video that shows two capsules side by side that animate using the ease-in ease-out animation. The capsule on the left is short, while the right-side capsule is tall. They animate for four seconds with the short capsule growing upwards to match the height of the tall capsule. Then the tall capsule shrinks to match the original height of the short capsule. For another four seconds, the capsule on the left shrinks to its original height, followed by the capsule on the right growing to its original height.")

///

/// - Parameter speed: The speed at which SwiftUI performs the animation.

/// - Returns: An animation with the adjusted speed.

```
public func speed(_ speed: Double) -> Animation}
```

```
@available(iOS 17.0, macOS 14.0, tvOS 17.0, watchOS 10.0, *)
```

```
extension Animation {
```

/// A persistent spring animation.

///

/// When mixed with other `spring()``

/// or `interactiveSpring()``

animations on the same property, each

```
    /// animation will be replaced by
    their successor, preserving
    /// velocity from one animation to
    the next. Optionally blends the
    /// duration values between springs
    over a time period.
    public static func spring(_ spring:
Spring, blendDuration: TimeInterval =
0.0) -> Animation

    /// An interpolating spring animation
    that uses a damped spring
    /// model to produce values in the
    range of one to zero.
    ///
    /// These values are used to
    interpolate within the `[from, to]` range
    /// of the animated
    /// property. Preserves velocity
    across overlapping animations by
    /// adding the effects of each
    animation.
    public static func
interpolatingSpring(_ spring: Spring,
initialVelocity: Double = 0.0) ->
Animation
}

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

    /// Causes the animation to report
```

```
logical completion after the specified
    /// duration, if it has not already
logically completed.
    ///
    /// Note that the indicated duration
will not cause the animation to
    /// continue running after the base
animation has fully completed.
    ///
    /// If the animation is removed
before the given duration is reached,
    /// logical completion will be
reported immediately.
    ///
    /// - Parameters:
    ///   - duration: The duration after
which the animation should report
    ///       that it is logically
complete.
    /// - Returns: An animation that
reports logical completion after the
    /// given duration.
public func logicallyComplete(after
duration: TimeInterval) -> Animation
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Animation {

    /// An interpolating spring animation
that uses a damped spring
    /// model to produce values in the
```

```
range [0, 1] that are then used
    /// to interpolate within the [from,
to] range of the animated
    /// property. Preserves velocity
across overlapping animations by
    /// adding the effects of each
animation.

    ///
    /// - Parameters:
    ///   - mass: The mass of the object
attached to the spring.
    ///   - stiffness: The stiffness of
the spring.
    ///   - damping: The spring damping
value.
    ///   - initialVelocity: the initial
velocity of the spring, as
    ///   a value in the range [0, 1]
representing the magnitude of
    ///   the value being animated.
    /// - Returns: a spring animation.
public static func
interpolatingSpring(mass: Double = 1.0,
stiffness: Double, damping: Double,
initialVelocity: Double = 0.0) ->
Animation

    /// An interpolating spring animation
that uses a damped spring
    /// model to produce values in the
range [0, 1] that are then used
    /// to interpolate within the [from,
to] range of the animated
```

```
    /// property. Preserves velocity
    across overlapping animations by
        /// adding the effects of each
    animation.

    ///
    /// - Parameters:
    ///   - duration: The perceptual
    duration, which defines the pace of the
        ///   spring. This is approximately
    equal to the settling duration, but
        ///   for very bouncy springs, will
    be the duration of the period of
        ///   oscillation for the spring.
    ///   - bounce: How bouncy the spring
should be. A value of 0 indicates
        ///   no bounces (a critically
    damped spring), positive values indicate
        ///   increasing amounts of
    bounciness up to a maximum of 1.0
        ///   (corresponding to undamped
    oscillation), and negative values
        ///   indicate overdamped springs
with a minimum value of -1.0.
    ///   - initialVelocity: the initial
velocity of the spring, as
        ///   a value in the range [0, 1]
representing the magnitude of
        ///   the value being animated.
    /// - Returns: a spring animation.

public static func
interpolatingSpring(duration:
TimeInterval = 0.5, bounce: Double = 0.0,
initialVelocity: Double = 0.0) ->
```

## Animation

```
    /// An interpolating spring animation  
    /// that uses a damped spring  
    /// model to produce values in the  
    /// range [0, 1] that are then used  
    /// to interpolate within the [from,  
    /// to] range of the animated  
    /// property. Preserves velocity  
    /// across overlapping animations by  
    /// adding the effects of each  
    /// animation.  
    ///  
    /// This uses the default parameter  
    /// values.  
    public static var  
interpolatingSpring: Animation { get }  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Animation {  
  
    /// Delays the start of the animation  
    /// by the specified number of seconds.  
    ///  
    /// Use this method to delay the  
    /// start of an animation. For example, the  
    /// following code animates the  
    /// height change of two capsules.  
    /// Animation of the first  
    ``Capsule`` begins immediately. However,  
    /// animation of the second one
```

doesn't begin until a half second later.

```
///
///    struct ContentView: View {
///        @State private var
adjustBy = 100.0
///
///        var body: some View {
///            VStack(spacing: 40) {
///                HStack(alignment:
.bottom) {
///                    Capsule()
///                    .frame(wi
dth: 50, height: 175 - adjustBy)
///                        .animatio
n(.easeInOut, value: adjustBy)
///                    Capsule()
///                    .frame(wi
dth: 50, height: 175 + adjustBy)
///                        .animatio
n(.easeInOut.delay(0.5), value: adjustBy)
///                }
///
///                Button("Animate")
{
///                    adjustBy *=
-1
///                }
///
///            }
///
///        }
///
///        @Video(source: "animation-15-
delay.mp4", poster: "animation-15-
```

delay.png", alt: "A video that shows two capsules side by side that animate using the ease-in ease-out animation. The capsule on the left is short, while the capsule on the right is tall. As they animate, the short capsule grows upwards to match the height of the tall capsule. Then the tall capsule shrinks to match the original height of the short capsule. Then the capsule on the left shrinks to its original height, followed by the capsule on the right growing to its original height.")

///

/// – Parameter delay: The number of seconds to delay the start of the /// animation.

/// – Returns: An animation with a delayed start.

```
public func delay(_ delay:  
TimeInterval) -> Animation  
{
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Animation {
```

/// A persistent spring animation.  
When mixed with other `spring()`  
/// or `interactiveSpring()`  
animations on the same property, each  
/// animation will be replaced by  
their successor, preserving

```
    /// velocity from one animation to
    the next. Optionally blends the
    /// duration values between springs
    over a time period.
    ///
    /// - Parameters:
    ///   - duration: The perceptual
    duration, which defines the pace of the
    ///   spring. This is approximately
    equal to the settling duration, but
    ///   for very bouncy springs, will
    be the duration of the period of
    ///   oscillation for the spring.
    ///   - bounce: How bouncy the spring
    should be. A value of 0 indicates
    ///   no bounces (a critically
    damped spring), positive values indicate
    ///   increasing amounts of
    bounciness up to a maximum of 1.0
    ///   (corresponding to undamped
    oscillation), and negative values
    ///   indicate overdamped springs
    with a minimum value of -1.0.
    ///   - blendDuration: The duration
    in seconds over which to
    ///   interpolate changes to the
    duration.
    /// - Returns: a spring animation.
public static func spring(duration:
TimeInterval = 0.5, bounce: Double = 0.0,
blendDuration: Double = 0) -> Animation

    /// A persistent spring animation.
```

When mixed with other `spring()``  
    /// or `interactiveSpring()``  
animations on the same property, each  
    /// animation will be replaced by  
their successor, preserving  
    /// velocity from one animation to  
the next. Optionally blends the  
    /// response values between springs  
over a time period.

    ///

    /// – Parameters:

    /// – response: The stiffness of  
the spring, defined as an  
    /// approximate duration in  
seconds. A value of zero requests  
    /// an infinitely-stiff spring,  
suitable for driving  
    /// interactive animations.  
    /// – dampingFraction: The amount  
of drag applied to the value  
    /// being animated, as a fraction  
of an estimate of amount  
    /// needed to produce critical  
damping.  
    /// – blendDuration: The duration  
in seconds over which to  
    /// interpolate changes to the  
response value of the spring.  
    /// – Returns: a spring animation.

**public static func** **spring**(**response**:  
**Double** = **0.5**, **dampingFraction**: **Double** =  
**0.825**, **blendDuration**: **TimeInterval** = **0**)  
–> **Animation**

```
    /// A persistent spring animation.  
When mixed with other `spring()``  
    /// or `interactiveSpring()``  
animations on the same property, each  
    /// animation will be replaced by  
their successor, preserving  
    /// velocity from one animation to  
the next. Optionally blends the  
    /// response values between springs  
over a time period.  
    ///  
    /// This uses the default parameter  
values.  
    public static var spring: Animation {  
get }  
  
    /// A convenience for a `spring`  
animation with a lower  
    /// `response` value, intended for  
driving interactive animations.  
    public static func  
interactiveSpring(response: Double =  
0.15, dampingFraction: Double = 0.86,  
blendDuration: TimeInterval = 0.25) ->  
Animation  
  
    /// A convenience for a `spring`  
animation with a lower  
    /// `duration` value, intended for  
driving interactive animations.  
    ///  
    /// This uses the default parameter
```

values.

```
    public static var interactiveSpring:  
Animation { get }  
  
        /// A convenience for a `spring`  
        animation with a lower  
        /// `response` value, intended for  
        driving interactive animations.  
        public static func  
interactiveSpring(duration: TimeInterval  
= 0.15, extraBounce: Double = 0.0,  
blendDuration: TimeInterval = 0.25) ->  
Animation  
  
        /// A smooth spring animation with a  
        predefined duration and no bounce.  
        public static var smooth: Animation {  
get }  
  
        /// A smooth spring animation with a  
        predefined duration and no bounce  
        /// that can be tuned.  
        ///  
        /// - Parameters:  
        /// - duration: The perceptual  
duration, which defines the pace of the  
        ///     spring. This is approximately  
equal to the settling duration, but  
        ///     for very bouncy springs, will  
be the duration of the period of  
        ///     oscillation for the spring.  
        /// - extraBounce: How much  
additional bounce should be added to the
```

```
base
    ///      bounce of 0.
    /// - blendDuration: The duration
in seconds over which to interpolate
    ///      changes to the duration.
    public static func smooth(duration:
TimeInterval = 0.5, extraBounce: Double =
0.0) -> Animation

        /// A spring animation with a
predefined duration and small amount of
        /// bounce that feels more snappy.
        public static var snappy: Animation {
get }

        /// A spring animation with a
predefined duration and small amount of
        /// bounce that feels more snappy and
can be tuned.
        ///
        /// - Parameters:
        /// - duration: The perceptual
duration, which defines the pace of the
        ///      spring. This is approximately
equal to the settling duration, but
        ///      for very bouncy springs, will
be the duration of the period of
        ///      oscillation for the spring.
        /// - extraBounce: How much
additional bounce should be added to the
base
        ///      bounce of 0.15.
        /// - blendDuration: The duration
```

```
in seconds over which to interpolate
    /// changes to the duration.
    public static func snappy(duration:
TimeInterval = 0.5, extraBounce: Double =
0.0) -> Animation

    /// A spring animation with a
predefined duration and higher amount of
    /// bounce.
    public static var bouncy: Animation {
get }

    /// A spring animation with a
predefined duration and higher amount of
    /// bounce that can be tuned.
    ///
    /// – Parameters:
    /// – duration: The perceptual
duration, which defines the pace of the
    /// spring. This is approximately
equal to the settling duration, but
    /// for very bouncy springs, will
be the duration of the period of
    /// oscillation for the spring.
    /// – extraBounce: How much
additional bounce should be added to the
base
    ///     bounce of 0.3.
    /// – blendDuration: The duration
in seconds over which to interpolate
    ///     changes to the duration.
    public static func bouncy(duration:
TimeInterval = 0.5, extraBounce: Double =
```

```
0.0) -> Animation
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
```

```
extension Animation {
```

```
    /// An animation with a specified  
duration that combines the behaviors of
```

```
    /// in and out easing animations.
```

```
    ///
```

```
    /// An easing animation provides  
motion with a natural feel by varying
```

```
    /// the acceleration and deceleration  
of the animation, which matches
```

```
    /// how things tend to move in  
reality. An ease in and out animation
```

```
    /// starts slowly, increasing its  
speed towards the halfway point, and
```

```
    /// finally decreasing the speed  
towards the end of the animation.
```

```
    ///
```

```
    /// Use `easeInOut(duration:)` when  
you want to specify the time it takes
```

```
    /// for the animation to complete.
```

```
Otherwise, use ``easeInOut`` to perform
```

```
    /// the animation for a default  
length of time.
```

```
    ///
```

```
    /// The following code shows an  
example of animating the size changes of
```

```
    /// a ``Circle`` using an ease in and  
out animation with a duration of
```

```
    /// one second.  
    ///  
    ///     struct ContentView: View {  
    ///         @State private var scale  
 = 0.5  
    ///  
    ///             var body: some View {  
    ///                 VStack {  
    ///                     Circle()  
    ///                         .scale(scale)  
    ///                         .animation(.e  
aseInOut(duration: 1.0), value: scale)  
    ///                     HStack {  
    ///                         Button["+"] {  
    scale += 0.1 }  
    ///                         Button["-"] {  
    scale -= 0.1 }  
    ///                     }  
    ///                 }  
    ///             }  
    ///         }  
    ///     }  
    ///  
    ///     @Video(source: "animation-13-  
easeineaseout-duration.mp4", poster:  
"animation-13-easeineaseout-  
duration.png", alt: "A video that shows a  
circle enlarging for one second, then  
shrinking for another second to its  
original size using an ease-in ease-out  
animation.")  
    ///  
    /// - Parameter duration: The length  
of time, expressed in seconds, that
```

```
    /// the animation takes to complete.  
    ///  
    /// – Returns: An ease-in ease-out  
    animation with a specified duration.  
    public static func  
easeInOut(duration: TimeInterval) ->  
Animation  
  
    /// An animation that combines the  
behaviors of in and out easing  
    /// animations.  
    ///  
    /// An easing animation provides  
motion with a natural feel by varying  
    /// the acceleration and deceleration  
of the animation, which matches  
    /// how things tend to move in  
reality. An ease in and out animation  
    /// starts slowly, increasing its  
speed towards the halfway point, and  
    /// finally decreasing the speed  
towards the end of the animation.  
    ///  
    /// The `easeInOut` animation has a  
default duration of 0.35 seconds. To  
    /// specify the duration, use the  
``easeInOut(duration:)`` method.  
    ///  
    /// The following code shows an  
example of animating the size changes of  
a  
    /// ``Circle`` using an ease in and  
out animation.
```

```
///  
///     struct ContentView: View {  
///         @State private var scale  
= 0.5  
///  
///             var body: some View {  
///                 VStack {  
///                     Circle()  
///                         .scale(scale)  
///                         .animation(.e  
aseInOut, value: scale)  
///                     HStack {  
///                         Button"+" {  
scale += 0.1 }  
///                         Button"-" {  
scale -= 0.1 }  
///                     }  
///                 }  
///             }  
///         }  
///     }  
/// @Video(source: "animation-12-  
easeineaseout.mp4", poster:  
"animation-12-easeineaseout.png", alt: "A  
video that shows a circle enlarging, then  
shrinking to its original size using an  
ease-in ease-out animation.")  
///  
/// - Returns: An ease-in ease-out  
animation with the default duration.  
public static var easeInOut:  
Animation { get }
```

```
    /// An animation with a specified
duration that starts slowly and then
    /// increases speed towards the end
of the movement.
    ///
    /// An easing animation provides
motion with a natural feel by varying
    /// the acceleration and deceleration
of the animation, which matches
    /// how things tend to move in
reality. With an ease in animation, the
    /// motion starts slowly and
increases its speed towards the end.
    ///
    /// Use `easeIn(duration:)` when you
want to specify the time it takes
    /// for the animation to complete.
Otherwise, use ``easeIn`` to perform the
    /// animation for a default length of
time.
    ///
    /// The following code shows an
example of animating the size changes of
    /// a ``Circle`` using an ease in
animation with a duration of one
    /// second.
    ///
    ///     struct ContentView: View {
    ///         @State private var scale
= 0.5
    ///
    ///             var body: some View {
    ///                 VStack {
```

```
    /**
     * An animation that starts slowly
     * and then increases speed towards the
     * end of the duration.
     */
    public static func easeIn(duration: TimeInterval) -> Animation {
        return Animation { value in
            let start = value / duration
            let end = 1 - start
            let scale = start * start * 3 + 1
            let circle = Circle()
                .scale(scale)
                .animation(.easeIn(duration: 1.0), value: scale)
            let stack = HStack {
                Button("+") {
                    scale += 0.1
                }
                Button("-") {
                    scale -= 0.1
                }
            }
            stack.animation(.easeIn(duration: 1.0))
            stack.modifier(ScalingEffect(value: scale))
            return circle
        }
    }
}
```

```
    /// end of the movement.  
    ///  
    /// An easing animation provides  
    motion with a natural feel by varying  
    /// the acceleration and deceleration  
    of the animation, which matches  
    /// how things tend to move in  
    reality. With an ease in animation, the  
    /// motion starts slowly and  
    increases its speed towards the end.  
    ///  
    /// The `easeIn` animation has a  
    default duration of 0.35 seconds. To  
    /// specify a different duration, use  
    ``easeIn(duration:)``.  
    ///  
    /// The following code shows an  
    example of animating the size changes of  
    /// a ``Circle`` using the ease in  
    animation.  
    ///  
    ///     struct ContentView: View {  
    ///         @State private var scale  
= 0.5  
    ///  
    ///         var body: some View {  
    ///             VStack {  
    ///                 Circle()  
    ///                     .scale(scale)  
    ///                     .animation(.e  
aseIn, value: scale)  
    ///             HStack {  
    ///                 Button"+" {
```



```
///  
/// Use `easeOut(duration:)` when you  
want to specify the time it takes  
/// for the animation to complete.  
Otherwise, use ``easeOut`` to perform  
/// the animation for a default  
length of time.  
///  
/// The following code shows an  
example of animating the size changes of  
/// a ``Circle`` using an ease out  
animation with a duration of one  
/// second.  
///  
/// struct ContentView: View {  
///     @State private var scale  
= 0.5  
///  
///     var body: some View {  
///         VStack {  
///             Circle()  
///                 .scale(scale)  
///                 .animation(.e  
aseOut(duration: 1.0), value: scale)  
///         HStack {  
///             Button["+"] {  
scale += 0.1 }  
///             Button["-"] {  
scale -= 0.1 }  
///         }  
///     }  
/// }
```

```
///  
/// @Video(source: "animation-09-  
easein-duration.mp4", poster:  
"animation-09-easein-duration.png", alt:  
"A video that shows a circle enlarging  
for one second, then shrinking for  
another second to its original size using  
an ease-in animation.")
```

```
///  
/// - Parameter duration: The length  
of time, expressed in seconds, that  
/// the animation takes to complete.  
///  
/// - Returns: An ease-out animation  
with a specified duration.
```

```
public static func easeOut(duration:  
TimeInterval) -> Animation
```

```
/// An animation that starts quickly  
and then slows towards the end of the  
/// movement.  
///  
/// An easing animation provides  
motion with a natural feel by varying  
/// the acceleration and deceleration  
of the animation, which matches  
/// how things tend to move in  
reality. With an ease out animation, the  
/// motion starts quickly and  
decreases its speed towards the end.
```

```
///  
/// The `easeOut` animation has a  
default duration of 0.35 seconds. To
```

```
    /// specify a different duration, use
``easeOut(duration:)``.
    ///
    /// The following code shows an
example of animating the size changes of
    /// a ``Circle`` using an ease out
animation.
    ///
    ///     struct ContentView: View {
    ///         @State private var scale
= 0.5
    ///
    ///         var body: some View {
    ///             VStack {
    ///                 Circle()
    ///                     .scale(scale)
    ///                     .animation(.e
aseOut, value: scale)
    ///             HStack {
    ///                 Button"+" {
scale += 0.1 }
    ///                 Button"-
" {
scale -= 0.1 }
    ///             }
    ///         }
    ///     }
    ///     @Video(source: "animation-10-
easeout.mp4", poster: "animation-10-
easeout.png", alt: "A video that shows a
circle enlarging, then shrinking to its
original size using an ease-out
```

```
animation.")

    /**
     * - Returns: An ease-out animation
     * with the default duration.
     */
    public static var easeOut: Animation
    { get }

    /**
     * An animation that moves at a
     * constant speed during a specified
     * duration.
    */

    /**
     * A linear animation provides a
     * mechanical feel to the motion because its
     * speed is consistent from start to
     * finish of the animation. This
     * constant speed makes a linear
     * animation ideal for animating the
     * movement of objects where changes
     * in the speed might feel awkward, such
     * as with an activity indicator.
    */

    /**
     * Use `linear(duration:)` when you
     * want to specify the time it takes
     * for the animation to complete.
     * Otherwise, use ``linear`` to perform the
     * animation for a default length of
     * time.
    */

    /**
     * The following code shows an
     * example of using linear animation with a
     * duration of two seconds to
     * animate the movement of a circle as it
     * moves
    */
```

```
    /// between the leading and trailing
    /// edges of the view. The color of the
    /// circle also animates from red to
    /// blue as it moves across the view.
    ///
    ///     struct ContentView: View {
    ///         @State private var
    isActive = false
    ///
    ///         var body: some View {
    ///             VStack(alignment:
    isActive ? .trailing : .leading) {
    ///                 Circle()
    ///                     .fill(isActiv
    e ? Color.red : Color.blue)
    ///                     .frame(width:
    50, height: 50)
    ///
    ///             Button("Animate")
    {
        ///
        /// withAnimation(.linear(duration: 2.0)) {
        ///
        isActive.toggle()
        ///
        ///         }
        ///
        ///         .frame(maxWidth:
    .infinity)
        ///
        ///         }
        ///
        ///     }
        ///
        ///     }
        ///
        ///     @Video(source: "animation-07-
```

```
linear-duration.mp4", poster:  
"animation-07-linear-duration.png", alt:  
"A video that shows a circle moving from  
the leading edge of the view to the  
trailing edge. The color of the circle  
also changes from red to blue as it moves  
across the view. Then the circle moves  
from the trailing edge back to the  
leading edge while also changing colors  
from blue to red.")
```

```
///
```

```
/// - Parameter duration: The length  
of time, expressed in seconds, that  
/// the animation takes to complete.
```

```
///
```

```
/// - Returns: A linear animation  
with a specified duration.
```

```
public static func linear(duration:  
TimeInterval) -> Animation
```

```
/// An animation that moves at a  
constant speed.
```

```
///
```

```
/// A linear animation provides a  
mechanical feel to the motion because its  
/// speed is consistent from start to  
finish of the animation. This
```

```
/// constant speed makes a linear  
animation ideal for animating the  
/// movement of objects where changes  
in the speed might feel awkward, such  
/// as with an activity indicator.
```

```
///
```

```
    /// The following code shows an
    /// example of using linear animation to
    /// animate the movement of a circle
    /// as it moves between the leading and
    /// trailing edges of the view. The
    /// circle also animates its color change
    /// as it moves across the view.
    ///
    /// struct ContentView: View {
    ///     @State private var
isActive = false
    ///
    ///         var body: some View {
    ///             VStack(alignment:
isActive ? .trailing : .leading) {
    ///                 Circle()
    ///                     .fill(isActiv
e ? Color.red : Color.blue)
    ///                     .frame(width:
50, height: 50)
    ///
    ///             Button("Animate")
{
    ///
withAnimation(.linear) {
    ///
isActive.toggle()
    ///
    ///
    ///
.infinity)
    ///
}
    ///
}
```

```
    /**
     */
    /**
     * @Video(source: "animation-06-linear.mp4", poster: "animation-06-linear.png", alt: "A video that shows a circle moving from the leading edge of the view to the trailing edge. The color of the circle also changes from red to blue as it moves across the view. Then the circle moves from the trailing edge back to the leading edge while also changing colors from blue to red.")
     */
    /**
     * The `linear` animation has a default duration of 0.35 seconds. To specify a different duration, use ``linear(duration:)``.
     */
    /**
     * - Returns: A linear animation with the default duration.
     */
    public static var linear: Animation {
        get }

        /**
         * An animation created from a cubic Bézier timing curve.
         */
        /**
         * Use this method to create a timing curve based on the control points of
         * a cubic Bézier curve. A cubic Bézier timing curve consists of a line
         * whose starting point is `(0, 0)` and whose end point is `(1, 1)`. Two
```

/// additional control points, `(p1x,  
    p1y)` and `(p2x, p2y)`, define the  
    /// shape of the curve.  
    ///  
    /// The slope of the line defines the  
    speed of the animation at that point  
    /// in time. A steep slopes causes  
    the animation to appear to run faster,  
    /// while a shallower slope appears  
    to run slower. The following  
    /// illustration shows a timing curve  
    where the animation starts and  
    /// finishes fast, but appears slower  
    through the middle section of the  
    /// animation.  
    ///  
    /// !-[An illustration of an XY graph  
    that shows the path of a Bézier timing  
    curve that an animation frame follows  
    over time. The horizontal x-axis has a  
    label with the text Time, and a label  
    with the text Frame appears along the  
    vertical y-axis. The path begins at the  
    graph's origin, labeled as (0.0, 0.0).  
    The path moves upwards, forming a concave  
    down shape. At the point of inflection,  
    the path continues upwards, forming a  
    concave up shape. A label with the text  
    First control point (p1x, p1y) appears  
    above the path. Extending from the label  
    is a dotted line pointing to the position  
    (0.1, 0.75) on the graph. Another label  
    with the text Second control point (p2x,

p2y) appears below the path. A dotted line extends from the label to the (0.85, 0.35) position on the graph.] (Animation-timingCurve-1)

```
///
/// The following code uses the
timing curve from the previous
/// illustration to animate a
``Circle`` as its size changes.
///
/// struct ContentView: View {
///     @State private var scale
= 1.0
///
///     var body: some View {
///         VStack {
///             Circle()
///                 .scaleEffect(
scale)
///                 .animation(
///                     .timingCu
rve(0.1, 0.75, 0.85, 0.35, duration:
2.0),
///                     value:
scale)
///         }
///         Button("Animate")
{
    ///         if scale ==
1.0 {
    ///             scale =
0.25
    ///         } else {
```

```
    /**
     * @param {Object} [options] - Options for the animation.
     * @param {number} [options.duration=1000] - Duration of the animation in milliseconds.
     * @param {string} [options.easing='ease-in-out'] - Easing function for the animation.
     * @param {number} [options.scale=1.0] - Scale factor for the animation.
     */
    function Animation(options) {
        const duration = options.duration || 1000;
        const easing = options.easing || 'ease-in-out';
        const scale = options.scale || 1.0;

        let startScale = 1.0;
        let endScale = 1.0;
        let startX = 0;
        let endX = 0;
        let startY = 0;
        let endY = 0;
        let p1x = 0;
        let p1y = 0;
        let p2x = 0;
        let p2y = 0;

        if (options.startScale) {
            startScale = options.startScale;
        }

        if (options.endScale) {
            endScale = options.endScale;
        }

        if (options.startX) {
            startX = options.startX;
        }

        if (options.endX) {
            endX = options.endX;
        }

        if (options.startY) {
            startY = options.startY;
        }

        if (options.endY) {
            endY = options.endY;
        }

        if (options.p1x) {
            p1x = options.p1x;
        }

        if (options.p1y) {
            p1y = options.p1y;
        }

        if (options.p2x) {
            p2x = options.p2x;
        }

        if (options.p2y) {
            p2y = options.p2y;
        }

        const timingCurve = [
            {t: 0, s: startScale},
            {t: 0.1, s: startScale}, // Control point 1
            {t: 0.85, s: endScale}, // Control point 2
            {t: 1.0, s: endScale}
        ];

        const x = startX + ((endX - startX) * timingCurve);
        const y = startY + ((endY - startY) * timingCurve);

        const durationString = duration + 'ms';

        const easingString = easing;

        const scaleString = scale + 'px';

        const styleString = `scale(${scale})`;

        const transformString = `translateX(${x}px) translateY(${y}px) ${styleString}`;

        const transitionString = `transition: ${durationString} ${easingString};`;

        const element = document.createElement('div');

        element.style = transitionString;

        element.innerHTML = `

/// @Video(source: "animation-14-timing-curve.mp4", poster: "animation-14-timing-curve.png", alt: "A video that shows a circle shrinking then growing to its original size using a timing curve animation. The first control point of the time curve is (0.1, 0.75) and the second is (0.85, 0.35).")



/// - Parameters:



/// - p1x: The x-coordinate of the first control point of the cubic Bézier curve.



/// - p1y: The y-coordinate of the first control point of the cubic Bézier curve.



/// - p2x: The x-coordinate of the second control point of the cubic Bézier curve.



/// - p2y: The y-coordinate of the second control point of the cubic Bézier curve.



/// - duration: The length of time, expressed in seconds, the animation takes to complete.


```

```
    /// - Returns: A cubic Bézier timing
    curve animation.
    public static func timingCurve(_ p1x:
Double, _ p1y: Double, _ p2x: Double, _ p2y: Double, duration: TimeInterval = 0.35) -> Animation
}

@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Animation {

    /// Repeats the animation for a
    specific number of times.
    ///
    /// Use this method to repeat the
    animation a specific number of times. For
    /// example, in the following code,
    the animation moves a truck from one
    /// edge of the view to the other
    edge. It repeats this animation three
    /// times.
    ///
    ///     struct ContentView: View {
    ///         @State private var
driveForward = true
    ///
    ///         private var
driveAnimation: Animation {
    ///             .easeInOut
    ///             .repeatCount(3,
autoreverses: true)
    ///             .speed(0.5)
```



```
///
/// The first time the animation
runs, the truck moves from the leading
/// edge to the trailing edge of the
view. The second time the animation
    /// runs, the truck moves from the
trailing edge to the leading edge
    /// because `autoreverse` is `true`.
If `autoreverse` were `false`, the
    /// truck would jump back to leading
edge before moving to the trailing
    /// edge. The third time the
animation runs, the truck moves from the
    /// leading to the trailing edge of
the view.
///
/// - Parameters:
///   - repeatCount: The number of
times that the animation repeats. Each
    /// repeated sequence starts at the
beginning when `autoreverse` is
    /// `false`.
///   - autoreverses: A Boolean value
that indicates whether the animation
    /// sequence plays in reverse after
playing forward. Autoreverse counts
    /// towards the `repeatCount`. For
instance, a `repeatCount` of one plays
    /// the animation forward once, but
it doesn't play in reverse even if
    /// `autoreverse` is `true`. When
`autoreverse` is `true` and
    /// `repeatCount` is `2`, the
```

animation moves forward, then reverses, then

```
    /// stops.  
    /// - Returns: An animation that  
    repeats for specific number of times.  
    public func repeatCount(_  
repeatCount: Int, autoreverses: Bool =  
true) -> Animation  
  
    /// Repeats the animation for the  
    lifespan of the view containing the  
    /// animation.  
    ///  
    /// Use this method to repeat the  
    animation until the instance of the view  
    /// no longer exists, or the view's  
    explicit or structural identity  
    /// changes. For example, the  
    following code continuously rotates a  
    /// gear symbol for the lifespan of  
    the view.  
    ///  
    /// struct ContentView: View {  
    ///     @State private var  
rotationDegrees = 0.0  
    ///  
    ///     private var animation:  
Animation {  
    ///         .linear  
    ///         .speed(0.1)  
    ///         .repeatForever(autore  
verses: false)  
    ///     }  
}
```

```
///  
///         var body: some View {  
///             Image(systemName:  
"gear")  
///                 .font(.system(siz  
e: 86))  
///                 .rotationEffect(.  
degrees(rotationDegrees))  
///                 .onAppear {  
///  
withAnimation(animation) {  
///  
rotationDegrees = 360.0  
///  
///  
///  
///  
///  
/// @Video(source: "animation-17-  
repeat-forever.mp4", poster:  
"animation-17-repeat-forever.png", alt:  
"A video that shows a gear that  
continuously rotates clockwise.")  
///  
/// - Parameter autoreverses: A  
Boolean value that indicates whether the  
/// animation sequence plays in  
reverse after playing forward.  
/// - Returns: An animation that  
continuously repeats.  
public func  
repeatForever(autoreverses: Bool = true)  
-> Animation
```

```
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Animation : Hashable {  
  
    /// Calculates the current value of  
    /// the animation.  
    /// - Returns: The current value of  
    /// the animation, or `nil` if the animation  
    /// has finished.  
    public func animate<V>(value: V,  
                           time: TimeInterval, context: inout  
                           AnimationContext<V>) -> V? where V :  
        VectorArithmetic  
  
    /// Calculates the current velocity  
    /// of the animation.  
    /// - Returns: The current velocity  
    /// of the animation, or `nil` if the the  
    /// velocity isn't available.  
    public func velocity<V>(value: V,  
                           time: TimeInterval, context:  
                           AnimationContext<V>) -> V? where V :  
        VectorArithmetic  
  
    /// Returns a Boolean value that  
    /// indicates whether the current animation  
    /// should merge with a previous  
    /// animation.  
    public func shouldMerge<V>(previous:
```

```
Animation, value: V, time: TimeInterval,  
context: inout AnimationContext<V>) ->  
Bool where V : VectorArithmetic  
  
    public var base: any CustomAnimation  
{ get }  
  
    /// 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)
```

```
    /// 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 }  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Animation :  
CustomStringConvertible,  
CustomDebugStringConvertible,  
CustomReflectable {  
  
    /// A textual representation of this  
    instance.  
    ///  
    /// Calling this property directly is  
    discouraged. Instead, convert an  
    /// instance of any type to a string  
    by using the `String(describing:)`
```

```
    /// initializer. This initializer
    works with any type, and uses the custom
    /// `description` property for types
    that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(describing: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
    /// `Point` type's `description`
property.
public var description: String {
get }

    /// A textual representation of this
instance, suitable for debugging.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
```

```
by using the `String(reflecting:)`  
    /// initializer. This initializer  
works with any type, and uses the custom  
    /// `debugDescription` property for  
types that conform to  
    /// `CustomDebugStringConvertible`:  
    ////  
    ///     struct Point:  
CustomDebugStringConvertible {  
    ///         let x: Int, y: Int  
    ////  
    ///         var debugDescription:  
String {  
    ///             return "(\(x), \(y))"  
    ///         }  
    ///     }  
    ////  
    ///     let p = Point(x: 21, y: 30)  
    ///     let s = String(reflecting: p)  
    ///     print(s)  
    ///     // Prints "(21, 30)"  
    ////  
    /// The conversion of `p` to a string  
in the assignment to `s` uses the  
    /// `Point` type's `debugDescription`  
property.  
    public var debugDescription: String {  
get }  
  
    /// The custom mirror for this  
instance.  
    ////  
    /// If this type has value semantics,
```

```
the mirror should be unaffected by
    /// subsequent mutations of the
instance.
    public var customMirror: Mirror { get
}
}

/// Contextual values that a custom
animation can use to manage state and
/// access a view's environment.
///
/// The system provides an
`AnimationContext` to a
``CustomAnimation`` instance
/// so that the animation can store and
retrieve values in an instance of
/// ``AnimationState``. To access these
values, use the context's
/// ``AnimationContext/state`` property.
///

/// For more convenient access to state,
create an ``AnimationStateKey`` and
/// extend `AnimationContext` to include
a computed property that gets and
/// sets the ``AnimationState`` value.
Then use this property instead of
/// ``AnimationContext/state`` to
retrieve the state of a custom animation.
For
/// example, the following code creates
an animation state key named
/// `PausableState`. Then the code
extends `AnimationContext` to include the
```

```
/// `pausableState` property:  
///  
///     private struct  
PausableState<Value: VectorArithmetic>:  
AnimationStateKey {  
    ///         var paused = false  
    ///         var pauseTime: TimeInterval =  
0.0  
    ///  
    ///         static var defaultValue: Self  
{ .init() }  
    ///     }  
    ///  
    ///     extension AnimationContext {  
    ///         fileprivate var  
pausableState: PausableState<Value> {  
    ///             get  
{ state[PausableState<Value>.self] }  
    ///             set  
{ state[PausableState<Value>.self] =  
newValue }  
    ///         }  
    ///     }  
    ///  
    /// To access the pausable state, the  
custom animation `PausableAnimation` uses  
    /// the `pausableState` property instead  
of the ``AnimationContext/state``  
    /// property:  
    ///  
    ///     struct PausableAnimation:  
CustomAnimation {  
    ///         let base: Animation
```

```
///  
///     func animate<V>(value: V,  
time: TimeInterval, context: inout  
AnimationContext<V>) -> V? where V :  
VectorArithmetic {  
///         let paused =  
context.environment.animationPaused  
///  
///         let pausableState =  
context.pausableState  
///         var pauseTime =  
pausableState.pauseTime  
///         if pausableState.paused !=  
= paused {  
///             pauseTime = time -  
pauseTime  
///             context.pausableState  
= PausableState(paused: paused,  
pauseTime: pauseTime)  
///         }  
///  
///         let effectiveTime =  
paused ? pauseTime : time - pauseTime  
///         let result =  
base.animate(value: value, time:  
effectiveTime, context: &context)  
///         return result  
///     }  
/// }  
///  
/// The animation can also retrieve  
environment values of the view that  
created
```

```
/// the animation. To retrieve a view's
environment value, use the context's
/// ``AnimationContext/environment``
property. For instance, the following
code
/// creates a custom
``EnvironmentValues`` property named
`animationPaused`, and the
/// view `PausableAnimationView` uses the
property to store the paused state:
///
///     extension EnvironmentValues {
///         @Entry var animationPaused:
Bool = false
///     }

///
///     struct PausableAnimationView:
View {
///         @State private var paused =
false
///

///         var body: some View {
///             VStack {
///                 ...
///             }
///             .environment(\.animationP
aused, paused)
///         }
///     }

///
/// Then the custom animation
`PausableAnimation` retrieves the paused
state
```

```
/// from the view's environment using the
``AnimationContext/environment``
/// property:
///
///     struct PausableAnimation:
CustomAnimation {
    func animate<V>(value: V,
time: TimeInterval, context: inout
AnimationContext<V>) -> V? where V :
VectorArithmetic {
        let paused =
context.environment.animationPaused
        ...
    }
}
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct AnimationContext<Value>
where Value : VectorArithmetic {

    /// The current state of a custom
animation.
    ///
    /// An instance of
``CustomAnimation`` uses this property to
read and
    /// write state values as the
animation runs.
    ///
    /// An alternative to using the
`state` property in a custom animation is
    /// to create an
``AnimationStateKey`` type and extend
```

```
``AnimationContext``
    /// with a custom property that
    returns the state as a custom type. For
        /// example, the following code
    creates a state key named
`PausableState`.
    /// It's convenient to store state
    values in the key type, so the
        /// `PausableState` structure
    includes properties for the stored state
        /// values `paused` and `pauseTime`.
    /**
     ///     private struct
PausableState<Value: VectorArithmetic>:
    AnimationStateKey {
    /**
         var paused = false
    /**
         var pauseTime:
TimeInterval = 0.0
    /**
         static var defaultValue:
Self { .init() }
    /**
}
    /**
        /// To provide access the pausable
state, the following code extends
        /// `AnimationContext` to include the
`pausableState` property. This
        /// property returns an instance of
the custom `PausableState` structure
        /// stored in
``AnimationContext/state``, and it can
also store a new
        /// `PausableState` instance in
```

```
`state`.  
    ///  
    ///     extension AnimationContext {  
    ///         fileprivate var  
pausableState: PausableState<Value> {  
    ///             get  
{ state[PausableState<Value>.self] }  
    ///             set  
{ state[PausableState<Value>.self] =  
newValue }  
    ///         }  
    ///  
    ///         /// Now a custom animation can use  
the `pausableState` property instead of  
    /// the ``AnimationContext/state``  
property as a convenient way to read and  
    /// write state values as the  
animation runs.  
    ///  
    ///     struct PausableAnimation:  
CustomAnimation {  
    ///         func animate<V>(value: V,  
time: TimeInterval, context: inout  
AnimationContext<V>) -> V? where V :  
VectorArithmetic {  
    ///             let pausableState =  
context.pausableState  
    ///             var pauseTime =  
pausableState.pauseTime  
    ///             ...  
    ///         }  
    ///     }
```

```
///  
public var state:  
AnimationState<Value>  
  
    /// Set this to `true` to indicate  
that an animation is logically complete.  
    ///  
    /// This controls when  
AnimationCompletionCriteria.logicallyComp  
lete  
    /// completion callbacks are fired.  
This should be set to `true` at most  
    /// once in the life of an animation,  
changing back to `false` later will be  
    /// ignored. If this is never set to  
`true`, the behavior is equivalent to  
    /// if this had been set to `true`  
just as the animation finished (by  
    /// returning `nil`).  
public var isLogicallyComplete: Bool  
  
    /// The current environment of the  
view that created the custom animation.  
    ///  
    /// An instance of  
``CustomAnimation`` uses this property to  
read  
    /// environment values from the view  
that created the animation. To learn  
    /// more about environment values  
including how to define custom  
    /// environment values, see  
``EnvironmentValues``.
```

```
    public var environment:  
        EnvironmentValues { get }  
  
        /// Creates a new context from  
        another one with a state that you  
        provide.  
        ///  
        /// Use this method to create a new  
        context that contains the state that  
        /// you provide and view environment  
        values from the original context.  
        ///  
        /// - Parameter state: The initial  
        state for the new context.  
        /// - Returns: A new context that  
        contains the specified state.  
    public func withState<T>(_ state:  
        AnimationState<T>) -> AnimationContext<T>  
    where T : VectorArithmetic  
}  
  
/// A container that stores the state for  
a custom animation.  
///  
/// An ``AnimationContext`` uses this  
type to store state for a  
/// ``CustomAnimation``. To retrieve the  
stored state of a context, you can  
/// use the ``AnimationContext/state``  
property. However, a more convenient  
/// way to access the animation state is  
to define an ``AnimationStateKey``  
/// and extend ``AnimationContext`` with
```

a computed property that gets  
/// and sets the animation state, as  
shown in the following code:

```
///
///     private struct
PausableState<Value: VectorArithmetic>:
AnimationStateKey {
    ///         static var defaultValue: Self
    { .init() }
    ///
    ///
    ///     extension AnimationContext {
    ///         fileprivate var
pausableState: PausableState<Value> {
    ///             get
    { state[PausableState<Value>.self] }
    ///             set
    { state[PausableState<Value>.self] =
newValue }
    ///
    }
    ///
    ///
    /// When creating an
``AnimationStateKey`` , it's convenient to
define the
/// state values that your custom
animation needs. For example, the
following
/// code adds the properties `paused` and
`pauseTime` to the `PausableState` .
/// animation state key:
///
///     private struct
```

```
PausableState<Value: VectorArithmetic>:  
AnimationStateKey {  
    ///         var paused = false  
    ///         var pauseTime: TimeInterval =  
0.0  
    ///  
    ///         static var defaultValue: Self  
{ .init() }  
    ///     }  
    ///  
    /// To access the pausable state in a  
`PausableAnimation`, the follow code  
/// calls `pausableState` instead of  
using the context's  
/// ``AnimationContext/state`` property.  
And because the animation state key  
/// `PausableState` defines properties  
for state values, the custom animation  
/// can read and write those values.  
///  
///     struct PausableAnimation:  
CustomAnimation {  
    ///         let base: Animation  
    ///  
    ///         func animate<V>(value: V,  
time: TimeInterval, context: inout  
AnimationContext<V>) -> V? where V :  
VectorArithmetric {  
    ///             let paused =  
context.environment.animationPaused  
    ///  
    ///             let pausableState =  
context.pausableState
```

```
///             var pauseTime =
pausableState.pauseTime
///             if pausableState.paused !
= paused {
///             pauseTime = time -
pauseTime
///             context.pausableState
= PausableState(paused: paused,
pauseTime: pauseTime)
///             }
///
///             let effectiveTime =
paused ? pauseTime : time - pauseTime
///             let result =
base.animate(value: value, time:
effectiveTime, context: &context)
///             return result
///             }
///
///             }
///
///             /**
///               Storing state for secondary
///               animations
///             */
///
///             A custom animation can also use
///             `AnimationState` to store the state of a
///             secondary animation. For example, the
///             following code creates an
///             ``AnimationStateKey`` that includes
///             the property `secondaryState`, which a
///             custom animation can use to store
///             other state:
///
///             private struct TargetState<Value:
```

```
VectorArithmetic>: AnimationStateKey {
    ///         var timeDelta = 0.0
    ///         var valueDelta = Value.zero
    ///         var secondaryState:
    AnimationState<Value>? = .init()
    ///
    ///         static var defaultValue: Self
    { .init() }
    ///
    ///
    ///         extension AnimationContext {
    ///             fileprivate var targetState:
    TargetState<Value> {
    ///                 get
    { state[TargetState<Value>.self] }
    ///                 set
    { state[TargetState<Value>.self] =
    newValue }
    ///
    ///         }
    ///
    ///         /// The custom animation
    `TargetAnimation` uses `TargetState` to
    store state
    /// data in `secondaryState` for another
    animation that runs as part of the
    /// target animation.
    ///
    ///         struct TargetAnimation:
    CustomAnimation {
    ///             var base: Animation
    ///             var secondary: Animation
    ///
```

```
///         func animate<V:  
VectorArithmetic>(value: V, time: Double,  
context: inout AnimationContext<V>) -> V?  
{  
    ///             var targetValue = value  
    ///             if let secondaryState =  
context.targetState.secondaryState {  
        ///                     var secondaryContext  
= context  
        ///  
secondaryContext.state = secondaryState  
        ///                     let secondaryValue =  
value - context.targetState.valueDelta  
        ///                     let result =  
secondary.animate(  
            ///                         value:  
secondaryValue, time: time -  
context.targetState.timeDelta,  
            ///                         context:  
&secondaryContext)  
        ///                     if let result =  
result {  
        ///  
context.targetState.secondaryState =  
secondaryContext.state  
        ///                     targetValue =  
result + context.targetState.valueDelta  
        ///                     } else {  
        ///  
context.targetState.secondaryState = nil  
        ///                     }  
        ///                     }  
        ///                     let result =
```

```
base.animate(value: targetValue, time:  
time, context: &context)  
/// if let result = result {  
///     targetValue = result  
/// } else if  
context.targetState.secondaryState == nil  
{  
///     return nil  
/// }  
/// return targetValue  
/// }  
///  
/// func shouldMerge<V:  
VectorArithmetic>(previous: Animation,  
value: V, time: Double, context: inout  
AnimationContext<V>) -> Bool {  
///     guard let previous =  
previous.base as? Self else { return  
false }  
///     var secondaryContext =  
context  
///     if let secondaryState =  
context.targetState.secondaryState {  
///         secondaryContext.state = secondaryState  
///         context.targetState.valueDelta =  
secondary.animate(  
///             value: value,  
time: time -  
context.targetState.timeDelta,  
///             context:  
&secondaryContext) ?? value
```

```
    } else {
    /**
context.targetState.valueDelta = value
    }
    // Reset the target each
time a merge occurs.
    /**
context.targetState.secondaryState
= .init()
    /**
context.targetState.timeDelta = time
    return base.shouldMerge(
    previous:
previous.base, value: value, time: time,
    context: &context)
    }
    /**
    }
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct AnimationState<Value> where
Value : VectorArithmetic {

    /**
Create an empty state container.
    /**
    /**
You don't typically create an
instance of ``AnimationState`` directly.
    /**
Instead, the ``AnimationContext``
provides the animation state to an
    /**
instance of ``CustomAnimation``.
public init()

    /**
Accesses the state for a custom
key.
```

```
///  
/// Create a custom animation state  
value by defining a key that conforms  
/// to the ``AnimationStateKey``  
protocol and provide the  
///  
``AnimationStateKey/defaultValue`` for  
the key. Also include properties  
/// to read and write state values  
that your ``CustomAnimation`` uses. For  
/// example, the following code  
defines a key named `PausableState` that  
/// has two state values, `paused`  
and `pauseTime`:  
///  
///     private struct  
PausableState<Value: VectorArithmetic>:  
AnimationStateKey {  
    ///         var paused = false  
    ///         var pauseTime:  
TimeInterval = 0.0  
    ///  
    ///             static var defaultValue:  
Self { .init() }  
    ///}  
    ///  
    /// Use that key with the subscript  
operator of the ``AnimationState``  
    /// structure to get and set a value  
for the key. For more convenient  
    /// access to the key value, extend  
``AnimationContext`` with a computed  
    /// property that gets and sets the
```

key's value.

```
///
///      extension AnimationContext {
///          fileprivate var
pausableState: PausableState<Value> {
    ///          get
{ state[PausableState<Value>.self] }
    ///          set
{ state[PausableState<Value>.self] =
newValue }
    ///      }
    ///
    ///
    /// To access the state values in a
``CustomAnimation``, call the custom
    /// computed property, then read and
write the state values that the
    /// custom ``AnimationStateKey``
provides.
///
///      struct PausableAnimation:
CustomAnimation {
    ///          let base: Animation
    ///
    ///          func animate<V>(value: V,
time: TimeInterval, context: inout
AnimationContext<V>) -> V? where V :
VectorArithmetic {
    ///          let paused =
context.environment.animationPaused
    ///
    ///          let pausableState =
context.pausableState
```

```
    /**
     * var pauseTime =
     * pausableState.pauseTime
     * if
     * pausableState.paused != paused {
     *   pauseTime = time
     * - pauseTime
     * }
     */
    context.pausableState =
    PausableState(paused: paused, pauseTime:
    pauseTime)
    /**
     */
    /**
     * let effectiveTime =
     * paused ? pauseTime : time - pauseTime
     * let result =
     * base.animate(value: value, time:
     * effectiveTime, context: &context)
     */
    return result
}
}

public subscript<K>(key: K.Type) ->
K.Value where K : AnimationStateKey
}
```

```
/// A key for accessing animation state
values.
///
/// To access animation state from an
``AnimationContext`` in a custom
/// animation, create an
`AnimationStateKey`. For example, the
following
/// code creates an animation state key
```

named `PausableState` and sets the  
/// value for the required  
``defaultValue`` property. The code also  
defines  
/// properties for state values that the  
custom animation needs when  
/// calculating animation values. Keeping  
the state values in the animation  
/// state key makes it more convenient to  
read and write those values in the  
/// implementation of a  
``CustomAnimation``.

```
///
///     private struct
PausableState<Value: VectorArithmetic>:
AnimationStateKey {
    ///         var paused = false
    ///         var pauseTime: TimeInterval =
0.0
    ///
    ///         static var defaultValue: Self
    { .init() }
    ///
}

///
/// To make accessing the value of the
animation state key more convenient,
/// define a property for it by extending
``AnimationContext``:
///

///     extension AnimationContext {
///         fileprivate var
pausableState: PausableState<Value> {
    ///
        get
}
```

```
{ state[PausableState<Value>.self] }
///           set
{ state[PausableState<Value>.self] =
newValue }
///           }
///           }
///           }

/// Then, you can read and write your
state in an instance of `CustomAnimation`
/// using the ``AnimationContext``:
///

///     struct PausableAnimation:
CustomAnimation {
    ///         let base: Animation
    ///
    ///         func animate<V>(value: V,
time: TimeInterval, context: inout
AnimationContext<V>) -> V? where V :
VectorArithmetic {
    ///             let paused =
context.environment.animationPaused
    ///
    ///             let pausableState =
context.pausableState
    ///             var pauseTime =
pausableState.pauseTime
    ///             if pausableState.paused !
= paused {
    ///                 pauseTime = time -
pauseTime
    ///                 context.pausableState
= PausableState(paused: paused,
pauseTime: pauseTime)
```

```
    /**
     * Returns the result of the animation
     * state key's base animation.
     */
    public func value(at time: TimeInterval) -> Value {
        let effectiveTime = paused ? pauseTime : time - pauseTime
        let result = base.animate(value: value, time: effectiveTime, context: &context)
        return result
    }
}

@available(iOS 17.0, macOS 14.0, tvOS 17.0, watchOS 10.0, *)
public protocol AnimationStateKey {

    /// The associated type representing
    /// the type of the animation state key's
    /// value.
    associatedtype Value

    /// The default value for the
    /// animation state key.
    static var defaultValue: Self.Value { get }

    /// A type-erased gesture.
    @available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
    @frozen public struct AnyGesture<Value> : Gesture {

        /// Creates an instance from another
        /// gesture.
    }
}
```

```
///  
/// - Parameter gesture: A gesture  
// that you use to create a new gesture.  
public init<T>(_ gesture: T) where  
Value == T.Value, T : Gesture  
  
    /// The type of gesture representing  
the body of `Self`.  
    @available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
    public typealias Body = Never  
}  
  
/// A color gradient.  
///  
/// When used as a ``ShapeStyle``, this  
type draws a linear gradient  
/// with start-point [0.5, 0] and end-  
point [0.5, 1].  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
@frozen public struct AnyGradient :  
Hashable, ShapeStyle, Sendable {  
  
    /// Creates a new instance from the  
specified gradient.  
    public init(_ gradient: Gradient)  
  
        /// 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 == (lhs:  
AnyGradient, rhs: AnyGradient) -> Bool  
  
        /// The type of shape style this will  
        resolve to.  
        ///  
        /// When you create a custom shape  
        style, Swift infers this type  
        /// from your implementation of the  
        required `resolve` function.  
        @available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
        public typealias Resolved = Never  
  
        /// 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 }  
    }
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension AnyGradient {  
  
    /// Returns a version of the gradient  
    /// that will use a specified  
    /// color space for interpolating  
    /// between its colors.  
    ///  
    ///  
    Rectangle().fill(.linearGradient(  
        ///          colors:  
        [.white, .blue]).colorSpace(.perceptual))  
    ///  
    /// - Parameters:  
    ///     - space: The color space the  
    new gradient will use to  
    ///         interpolate its constituent  
    colors.  
    ///  
    /// - Returns: A new gradient that  
    interpolates its colors in the  
    ///     specified color space.  
    ///  
    public func colorSpace(_ space:  
Gradient.ColorSpace) -> AnyGradient  
}  
  
/// A type-erased instance of the layout  
protocol.  
///  
/// Use an `AnyLayout` instance to enable  
dynamically changing the
```

```
/// type of a layout container without
/// destroying the state of the subviews.
/// For example, you can create a layout
/// that changes between horizontal and
/// vertical layouts based on the current
Dynamic Type setting:
///
///     struct DynamicLayoutExample: View
{
///
@Environment(\.dynamicTypeSize) var
dynamicTypeSize
///
///         var body: some View {
///             let layout =
dynamicTypeSize <= .medium ?
///
AnyLayout(HStackLayout()) :
AnyLayout(VStackLayout())
///
///             layout {
///                 Text("First label")
///                 Text("Second label")
///             }
///         }
///
///     }
///
/// The types that you use with
`AnyLayout` must conform to the
``Layout``
/// protocol. The above example chooses
between the ``HStackLayout`` and
/// ``VStackLayout`` types, which are
```

versions of the built-in ``HStack``  
/// and ``VStack`` containers that  
conform to the protocol. You can also  
/// use custom layout types that you  
define.

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
@frozen public struct AnyLayout : Layout  
{
```

    /// Creates a type-erased value that  
wraps the specified layout.

    ///

    /// You can switch between type-  
erased layouts without losing the state  
    /// of the subviews.

```
    public init<L>(_ layout: L) where L :  
Layout
```

    /// Cached values associated with the  
layout instance.

    ///

    /// If you create a cache for your  
custom layout, you can use

    /// a type alias to define this type  
as your data storage type.

    /// Alternatively, you can refer to  
the data storage type directly in all

    /// the places where you work with  
the cache.

    ///

    /// See ``makeCache(subviews:)`` for  
more information.

```
public struct Cache {  
}  
  
    /// The type defining the data to  
    // animate.  
    public typealias AnimatableData  
  
        /// Creates and initializes a cache  
        // for a layout instance.  
        ///  
        /// You can optionally use a cache to  
        // preserve calculated values across  
        /// calls to a layout container's  
        // methods. Many layout types don't need  
        /// a cache, because SwiftUI  
        // automatically reuses both the results of  
        /// calls into the layout and the  
        // values that the layout reads from its  
        /// subviews. Rely on the protocol's  
        // default implementation of this method  
        /// if you don't need a cache.  
        ///  
        /// However you might find a cache  
        // useful when:  
        ///  
        /// - The layout container repeats  
        // complex, intermediate calculations  
        /// across calls like  
``sizeThatFits(proposal:subviews:cache:)``  
,  
        ///  
``placeSubviews(in:proposal:subviews:cache:)``, and
```

```
    /**
     ``explicitAlignment(of:in:proposal:subviews:cache:)``.
    /// You might be able to improve
    performance by calculating values
    /// once and storing them in a cache.
    /// – The layout container reads many
    ``LayoutValueKey`` values from
    /// subviews. It might be more
    efficient to do that once and store the
    /// results in the cache, rather than
    rereading the subviews' values before
    /// each layout call.
    /// – You want to maintain working
    storage, like temporary Swift arrays,
    /// across calls into the layout, to
    minimize the number of allocation
    /// events.
    /**
     /// Only implement a cache if
     profiling shows that it improves
     performance.
    /**
    /// Initialize a cache
    /**
     /// Implement the
    `makeCache(subviews:)` method to create a
    cache.
    /// You can add computed values to
    the cache right away, using information
    /// from the `subviews` input
    parameter, or you can do that later. The
    /// methods of the ``Layout``
```

```
protocol that can access the cache
    /// take the cache as an in-out
parameter, which enables you to modify
    /// the cache anywhere that you can
read it.

    ///
    /// You can use any storage type that
makes sense for your layout
    /// algorithm, but be sure that you
only store data that you derive
    /// from the layout and its subviews
(lazily, if possible). For this to
    /// work correctly, SwiftUI needs to
be able to call this method to
    /// recreate the cache without
changing the layout result.

    ///
    /// When you return a cache from this
method, you implicitly define a type
    /// for your cache. Be sure to either
make the type of the `cache`
    /// parameters on your other
``Layout`` protocol methods match, or use
    /// a type alias to define the
``Cache`` associated type.

    ///
    /// ### Update the cache
    ///
    /// If the layout container or any of
its subviews change, SwiftUI
    /// calls the
``updateCache(_:subviews:)`` method so
you can
```

```
    /// modify or invalidate the contents  
    of the  
        /// cache. The default implementation  
        of that method calls the  
            /// `makeCache(subviews:)` method to  
            recreate the cache, but you can  
                /// provide your own implementation  
                of the update method to take an  
                    /// incremental approach, if  
                    appropriate.  
                    ///  
                    /// - Parameters:  
                    ///     - subviews: A collection of  
                    proxy instances that represent the  
                        /// views that the container  
                        arranges. You can use the proxies in the  
                            /// collection to get information  
                            about the subviews as you  
                                /// calculate values to store in  
                                the cache.  
                                ///  
                                /// - Returns: Storage for calculated  
                                data that you share among  
                                    /// the methods of your custom  
                                    layout container.  
public func makeCache(subviews:  
AnyLayout.Subviews) -> AnyLayout.Cache  
  
    /// Updates the layout's cache when  
    something changes.  
    ///  
    /// If your custom layout container  
    creates a cache by implementing the
```

```
    /// ``makeCache(subviews:)`` method,  
SwiftUI calls the update method  
    /// when your layout or its subviews  
change, giving you an opportunity  
    /// to modify or invalidate the  
contents of the cache.  
    /// The method's default  
implementation recreates the  
    /// cache by calling the  
``makeCache(subviews:)`` method,  
    /// but you can provide your own  
implementation to take an  
    /// incremental approach, if  
appropriate.  
    ///  
    /// - Parameters:  
    /// - cache: Storage for calculated  
data that you share among  
    /// the methods of your custom  
layout container.  
    /// - subviews: A collection of  
proxy instances that represent the  
    /// views arranged by the  
container. You can use the proxies in the  
    /// collection to get information  
about the subviews as you  
    /// calculate values to store in  
the cache.  
    public func updateCache(_ cache:  
inout AnyLayout.Cache, subviews:  
AnyLayout.Subviews)  
  
    /// Returns the preferred spacing
```

values of the composite view.

```
///
/// Implement this method to provide
custom spacing preferences
/// for a layout container. The value
you return affects
/// the spacing around the container,
but it doesn't affect how the
/// container arranges subviews
relative to one another inside the
/// container.
///
/// Create a custom ``ViewSpacing``
instance for your container by
/// initializing one with default
values, and then merging that with
/// spacing instances of certain
subviews. For example, if you define
/// a basic vertical stack that
places subviews in a column, you could
/// use the spacing preferences of
the subview edges that make
/// contact with the container's
edges:
///
/// extension BasicVStack {
///     func spacing(subviews:
Subviews, cache: inout ()) -> ViewSpacing
{
    ///
    var spacing =
ViewSpacing()
    ///
    /// for index in
```

```
subviews.indices {  
    /// var edges:  
    Edge.Set = [.leading, .trailing]  
    /// if index == 0  
    { edges.formUnion(.top) }  
    /// if index ==  
    subviews.count - 1  
    { edges.formUnion(.bottom) }  
    ///  
    spacing.formUnion(subviews[index].spacing  
, edges: edges)  
    ///}  
    ///  
    /// return spacing  
    ///}  
    ///}  
    ///  
    /// In the above example, the first  
and last subviews contribute to the  
/// spacing above and below the  
container, respectively, while all  
subviews  
    /// affect the spacing on the leading  
and trailing edges.  
    ///  
    /// If you don't implement this  
method, the protocol provides a default  
    /// implementation that merges the  
spacing preferences across all subviews  
on all edges.  
    ///  
    /// - Parameters:  
    ///     - subviews: A collection of
```

```
proxy instances that represent the
    ///      views that the container
arranges. You can use the proxies in the
    ///      collection to get information
about the subviews as you determine
    ///      how much spacing the
container prefers around it.
    /// - cache: Optional storage for
calculated data that you can share among
    ///      the methods of your custom
layout container. See
    ///      ``makeCache(subviews:)`` for
details.

    ///
    /// - Returns: A ``ViewSpacing``
instance that describes the preferred
    ///      spacing around the container
view.



public func spacing(subviews:



AnyLayout.Subviews, cache: inout



AnyLayout.Cache) -> ViewSpacing



/// Returns the size of the composite
view, given a proposed size
    /// and the view's subviews.



///
    /// Implement this method to tell
your custom layout container's parent
    /// view how much space the container
needs for a set of subviews, given
    /// a size proposal. The parent might
call this method more than once
    /// during a layout pass with


```

different proposed sizes to test the  
flexibility of the container,  
using proposals like:

```
///
/// * The ``ProposedViewSize/zero`` proposal; respond with the
///   layout's minimum size.
/// * The
``ProposedViewSize/infinity`` proposal;
respond with the
///   layout's maximum size.
/// * The
``ProposedViewSize/unspecified`` proposal; respond with the
///   layout's ideal size.
///
/// The parent might also choose to
test flexibility in one dimension at a
/// time. For example, a horizontal
stack might propose a fixed height and
/// an infinite width, and then the
same height with a zero width.
///
/// The following example calculates
the size for a basic vertical stack
/// that places views in a column,
with no spacing between the views:
///
///     private struct BasicVStack:
Layout {
    ///
    func sizeThatFits(
        ///
        proposal:
ProposedViewSize,
```

```
    /**
     * - (CGSize) layoutSubviews:(Subviews *)subviews
     *   cache:(inout CGSize *)cache
     */
    - (CGSize) layoutSubviews:(Subviews *)subviews
        cache:(inout CGSize *)cache {
        // ...
        CGSize result = CGSizeMake(0, 0);
        for (Subview *subview in subviews) {
            CGSize size = subview.sizeThatFits(.unspecified);
            if (size.width > result.width) {
                result.width = size.width;
            }
            result.height += size.height;
        }
        *cache = result;
        return result;
    }

    /**
     * - (void) placeSubviews:(Subviews *)subviews
     */
    - (void) placeSubviews:(Subviews *)subviews {
        // ...
        for (Subview *subview in subviews) {
            subview.placeSubviews();
        }
    }
}

// This layout also needs
// a placeSubviews() implementation.
// ...
// The implementation asks each
// subview for its ideal size by calling the
// ``LayoutSubview/sizeThatFits(_:)`` method
// with an
// ``ProposedViewSize/unspecified`` proposed size.
// It then reduces these values into
// a single size that represents
// the maximum subview width and the
// sum of subview heights.
// Because this example isn't
// flexible, it ignores its size proposal
```

```
    /// input and always returns the same
    value for a given set of subviews.
    /**
     /// SwiftUI views choose their own
     size, so the layout engine always
     /// uses a value that you return from
     this method as the actual size of the
     /// composite view. That size factors
     into the construction of the `bounds`
     /// input to the
``placeSubviews(in:proposal:subviews:cach
e:)`` method.
    /**
     /// - Parameters:
     ///   - proposal: A size proposal for
     the container. The container's parent
     ///   view that calls this method
     might call the method more than once
     ///   with different proposals to
     learn more about the container's
     ///   flexibility before deciding
     which proposal to use for placement.
     ///   - subviews: A collection of
     proxies that represent the
     ///   views that the container
     arranges. You can use the proxies in the
     ///   collection to get information
     about the subviews as you determine
     ///   how much space the container
     needs to display them.
     ///   - cache: Optional storage for
     calculated data that you can share among
     ///   the methods of your custom
```

```
layout container. See
    ///      ``makeCache(subviews:)`` for
details.
    ///
    /// - Returns: A size that indicates
how much space the container
    /// needs to arrange its subviews.
public func sizeThatFits(proposal:
ProposedViewSize, subviews:
AnyLayout.Subviews, cache: inout
AnyLayout.Cache) -> CGSize
```

```
    /// Assigns positions to each of the
layout's subviews.
    ///
    /// SwiftUI calls your implementation
of this method to tell your
    /// custom layout container to place
its subviews. From this method, call
    /// the
``LayoutSubview/place(at:anchor:proposal:
)` ` method on each
    /// element in `subviews` to tell the
subviews where to appear in the
    /// user interface.
    ///
    /// For example, you can create a
basic vertical stack that places views
    /// in a column, with views
horizontally aligned on their leading
edge:
    ///
    ///     struct BasicVStack: Layout {
```

```
/// func placeSubviews(  
///   in bounds: CGRect,  
///   proposal:  
ProposedViewSize,  
///   subviews: Subviews,  
///   cache: inout ())  
/// ) {  
///   var point =  
bounds.origin  
///   for subview in  
subviews {  
///     subview.place(at:  
point, anchor: .topLeading,  
proposal: .unspecified)  
///     point.y +=  
subview.dimensions(in: .unspecified).heig  
ht  
///   }  
/// }  
/// // This layout also needs  
a sizeThatFits() implementation.  
/// }  
///  
/// The example creates a placement  
point that starts at the origin of the  
/// specified `bounds` input and uses  
that to place the first subview. It  
/// then moves the point in the y  
dimension by the subview's height,  
/// which it reads using the  
``LayoutSubview/dimensions(in:)`` method.  
/// This prepares the point for the
```



```
    /// The spacing's
``ViewSpacing/distance(to:along:)``
method considers the
    /// preferences of adjacent views on
the edge where they meet. It returns
    /// the smallest distance that
satisfies both views' preferences for the
    /// given edge. For example, if one
view prefers at least `2` points on its
    /// bottom edge, and the next view
prefers at least `8` points on its top
    /// edge, the distance method returns
`8`, because that's the smallest
    /// value that satisfies both
preferences.

    ///
    /// Update the placement calculations
to use the spacing values:
    ///
    ///     var point = bounds.origin
    ///     for (index, subview) in
subviews.enumerated() {
    ///         if index > 0 { point.y +=
spacing[index - 1] } // Add spacing.
    ///         subview.place(at: point,
anchor: .topLeading,
proposal: .unspecified)
    ///         point.y +=
subview.dimensions(in: .unspecified).heig
ht
    ///
    ///
    /// Be sure that you use computations
```

during placement that are consistent  
    /// with those in your implementation  
of other protocol methods for a given  
    /// set of inputs. For example, if  
you add spacing during placement,  
    /// make sure your implementation of  
    ///  
``sizeThatFits(proposal:subviews:cache:)``  
` accounts for the extra space.  
    /// Similarly, if the sizing method  
returns different values for different  
    /// size proposals, make sure the  
placement method responds to its  
    /// `proposal` input in the same way.  
    ///  
    /// - Parameters:  
    /// - bounds: The region that the  
container view's parent allocates to the  
    /// container view, specified in  
the parent's coordinate space.  
    /// Place all the container's  
subviews within the region.  
    /// The size of this region  
matches a size that your container  
    /// previously returned from a  
call to the  
    ///  
``sizeThatFits(proposal:subviews:cache:)``  
` method.  
    /// - proposal: The size proposal  
from which the container generated the  
    /// size that the parent used to  
create the `bounds` parameter.

```
    ///      The parent might propose more  
    than one size before calling the  
    ///      placement method, but it  
    always uses one of the proposals and the  
    ///      corresponding returned size  
    when placing the container.
```

```
    /// - subviews: A collection of  
    proxies that represent the  
    ///      views that the container  
    arranges. Use the proxies in the  
    collection
```

```
    ///      to get information about the  
    subviews and to tell the subviews  
    ///      where to appear.
```

```
    /// - cache: Optional storage for  
    calculated data that you can share among
```

```
    ///      the methods of your custom  
    layout container. See
```

```
    ///      ``makeCache(subviews:)`` for  
    details.
```

```
public func placeSubviews(in bounds:  
    CGRect, proposal: ProposedViewSize,  
    subviews: AnyLayout.Subviews, cache:  
    inout AnyLayout.Cache)
```

```
    /// Returns the position of the  
    specified horizontal alignment guide  
    along
```

```
    /// the x axis.
```

```
    ///
```

```
    /// Implement this method to return a  
    value for the specified alignment
```

```
    /// guide of a custom layout
```

container. The value you return affects  
    /// the placement of the container as  
a whole, but it doesn't affect how the  
    /// container arranges subviews  
relative to one another.

    ///  
    /// You can use this method to put an  
alignment guide in a nonstandard  
    /// position. For example, you can  
indent the container's leading edge  
    /// alignment guide by 10 points:

    ///  
    /// extension BasicVStack {  
    /// func explicitAlignment(  
    /// of guide:  
HorizontalAlignment,  
    /// in bounds: CGRect,  
    /// proposal:  
ProposedViewSize,  
    /// subviews: Subviews,  
    /// cache: inout ())  
    /// ) -> CGFloat? {  
    /// if guide == .leading  
{  
    /// return  
bounds minX + 10  
    /// }  
    /// return nil  
    /// }  
    /// }  
    ///  
    /// The above example returns `nil`  
for other guides to indicate that they

```
    /// don't have an explicit value. A
guide without an explicit value behaves
    /// as it would for any other view.
If you don't implement the
    /// method, the protocol's default
implementation merges the
    /// subviews' guides.
    ///
    /// - Parameters:
    ///   - guide: The
``HorizontalAlignment`` guide that the
method calculates
    ///   the position of.
    ///   - bounds: The region that the
container view's parent allocates to the
    ///   container view, specified in
the parent's coordinate space.
    ///   - proposal: A proposed size for
the container.
    ///   - subviews: A collection of
proxy instances that represent the
    ///   views arranged by the
container. You can use the proxies in the
    ///   collection to get information
about the subviews as you determine
    ///   where to place the guide.
    ///   - cache: Optional storage for
calculated data that you can share among
    ///   the methods of your custom
layout container. See
    ///   ``makeCache(subviews:)`` for
details.
    ///
```

```
    /// - Returns: The guide's position  
    relative to the `bounds`.  
    /// Return `nil` to indicate that  
    the guide doesn't have an explicit  
    /// value.  
    public func explicitAlignment(of  
guide: HorizontalAlignment, in bounds:  
CGRect, proposal: ProposedViewSize,  
subviews: AnyLayoutSubviews, cache:  
inout AnyLayoutCache) -> CGFloat?  
  
    /// Returns the position of the  
    specified vertical alignment guide along  
    /// the y axis.  
    ///  
    /// Implement this method to return a  
    value for the specified alignment  
    /// guide of a custom layout  
    container. The value you return affects  
    /// the placement of the container as  
    a whole, but it doesn't affect how the  
    /// container arranges subviews  
    relative to one another.  
    ///  
    /// You can use this method to put an  
    alignment guide in a nonstandard  
    /// position. For example, you can  
    raise the container's bottom edge  
    /// alignment guide by 10 points:  
    ///  
    ///     extension BasicVStack {  
    ///         func explicitAlignment(  
    ///             of guide:
```



the parent's coordinate space.

    /// – proposal: A proposed size for the container.

    /// – subviews: A collection of proxy instances that represent the

        /// views arranged by the container. You can use the proxies in the

        /// collection to get information about the subviews as you determine

        /// where to place the guide.

    /// – cache: Optional storage for calculated data that you can share among

        /// the methods of your custom layout container. See

    ///     ``makeCache(subviews:)`` for details.

    ///

    /// – Returns: The guide's position relative to the `bounds`.

    /// Return `nil` to indicate that the guide doesn't have an explicit

        /// value.

    public func explicitAlignment(of guide: VerticalAlignment, in bounds: CGRect, proposal: ProposedViewSize, subviews: AnyLayout.Subviews, cache: inout AnyLayout.Cache) -> CGFloat?

    /// The data to animate.

    public var animatableData: AnyLayout.AnimatableData  
{}

```
/// The base type of type-erased
locations with value-type Value.
///
/// It is annotated as `@unchecked
Sendable` so that user types such as
/// `State`, and `SceneStorage` can be
cleanly `Sendable`. However, it is
/// also the user types' responsibility
to ensure that `get`, and `set` does
/// not access the graph concurrently
(`get` should not be called while graph
/// is updating, for example).
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
open class AnyLocation<Value> :
AnyLocationBase, @unchecked Sendable {
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension AnyLocation : Equatable {

    /// 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 == (lhs:  
AnyLocation<Value>, rhs:  
AnyLocation<Value>) -> Bool  
}
```

```
/// The base type of all type-erased  
locations.
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
open class AnyLocationBase {  
}
```

```
/// A type-erased shape value.
```

```
///
```

```
/// You can use this type to dynamically  
switch between shape types:
```

```
///
```

```
///     struct MyClippedView: View {  
///         var isCircular: Bool  
///  
///         var body: some View {  
///             OtherView().clipShape(isCircular ?  
///                                         AnyShape(Circle()) :  
///                                         AnyShape(Capsule()))  
///         }  
///     }
```

```
///
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
@frozen public struct AnyShape : Shape,  
@unchecked Sendable {
```

```
    /// Create an any shape instance from  
a shape.
```

```
    public init<S>(_ shape: S) where S :  
Shape
```

```
    /// Describes this shape as a path  
within a rectangular frame of reference.
```

```
    ///
```

```
    /// - Parameter rect: The frame of  
reference for describing this shape.
```

```
    ///
```

```
    /// - Returns: A path that describes  
this shape.
```

```
nonisolated public func path(in rect:  
CGRect) -> Path
```

```
    /// Returns the size of the view that  
will render the shape, given
```

```
    /// a proposed size.
```

```
    ///
```

```
    /// Implement this method to tell the  
container of the shape how
```

```
    /// much space the shape needs to  
render itself, given a size
```

```
    /// proposal.
```

```
    ///
```

```
    /// See
```

```
``Layout/sizeThatFits(proposal:subviews:c  
ache:)``
```

```
    /// for more details about how the  
layout system chooses the size of
```

```
    /// views.
```

```
///  
/// - Parameters:  
///   - proposal: A size proposal for  
the container.  
///  
/// - Returns: A size that indicates  
how much space the shape needs.  
nonisolated public func  
sizeThatFits(_ proposal:  
ProposedViewSize) -> CGSize  
  
    /// The type defining the data to  
animate.  
    public typealias AnimatableData  
  
    /// The data to animate.  
    public var animatableData:  
AnyShape.AnimatableData  
  
    /// The type of view representing the  
body of this view.  
    ///  
    /// When you create a custom view,  
Swift infers this type from your  
    /// implementation of the required  
``View/body-swift.property`` property.  
    @available(iOS 16.0, tvOS 16.0,  
watchOS 9.0, macOS 13.0, *)  
    public typealias Body  
}  
  
    /// A type-erased ShapeStyle value.  
@available(iOS 15.0, macOS 12.0, tvOS
```

```
15.0, watchOS 8.0, *)
@frozen public struct AnyShapeStyle : ShapeStyle {

    /// Create an instance from `style`.
    public init<S>(_ style: S) where S : ShapeStyle

        /// The type of shape style this will resolve to.
        ///
        /// When you create a custom shape style, Swift infers this type
        /// from your implementation of the required `resolve` function.
        @available(iOS 17.0, tvOS 17.0,
        watchOS 10.0, macOS 14.0, *)
        public typealias Resolved = Never
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension AnyShapeStyle.Storage : @unchecked Sendable {

}

/// A type-erased transition.
///
/// - See Also: `Transition`
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct AnyTransition {
```

```
    /// Create an instance that type-
    erases `transition`.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public init<T>(_ transition: T) where
T : Transition
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension AnyTransition {

    /// Returns a transition defined
    between an active modifier and an
    identity
    /// modifier.
    public static func
modifier<E>(active: E, identity: E) ->
AnyTransition where E : ViewModifier
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension AnyTransition {

    public static func offset(_ offset:
CGSize) -> AnyTransition

        public static func offset(x: CGFloat
= 0, y: CGFloat = 0) -> AnyTransition
}

@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
extension AnyTransition {

    /// Returns a transition that moves
    the view away, towards the specified
    /// edge of the view.
    public static func move(edge: Edge)
-> AnyTransition
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension AnyTransition {

    /// A transition from transparent to
    opaque on insertion, and from opaque to
    /// transparent on removal.
    public static let opacity:
AnyTransition
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension AnyTransition {

    /// Provides a composite transition
    that uses a different transition for
    /// insertion versus removal.
    public static func
asymmetric(insertion: AnyTransition,
removal: AnyTransition) -> AnyTransition
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension AnyTransition {  
  
    /// A transition that inserts by  
    moving in from the leading edge, and  
    /// removes by moving out towards the  
    trailing edge.  
    ///  
    /// - SeeAlso:  
    `AnyTransition.move(edge:)`  
    public static var slide:  
        AnyTransition { get }  
}  
  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension AnyTransition {  
  
    /// Creates a transition that when  
    added to a view will animate the  
    /// view's insertion by moving it in  
    from the specified edge while  
    /// fading it in, and animate its  
    removal by moving it out towards  
    /// the opposite edge and fading it  
    out.  
    ///  
    /// - Parameters:  
    ///     - edge: the edge from which the  
    view will be animated in.  
    ///  
    /// - Returns: A transition that
```

```
animates a view by moving and
    /// fading it.
    public static func push(from edge:
Edge) -> AnyTransition
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension AnyTransition {

    /// Attaches an animation to this
transition.
    public func animation(_ animation:
Animation?) -> AnyTransition
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension AnyTransition {

    /// Combines this transition with
another, returning a new transition that
    /// is the result of both transitions
being applied.
    public func combined(with other:
AnyTransition) -> AnyTransition
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension AnyTransition {

    /// A transition that returns the
```

```
input view, unmodified, as the output
    /// view.
    public static let identity:
AnyTransition
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension AnyTransition {

    /// Returns a transition that scales
    the view.
    public static var scale:
AnyTransition { get }

    /// Returns a transition that scales
    the view by the specified amount.
    public static func scale(scale:
CGFloat, anchor: UnitPoint = .center) ->
AnyTransition
}

/// A type-erased view.
///
/// An `AnyView` allows changing the type
/// of view used in a given view
/// hierarchy. Whenever the type of view
/// used with an `AnyView` changes, the old
/// hierarchy is destroyed and a new
/// hierarchy is created for the new type.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct AnyView : View {
```

```
    /// Create an instance that type-
    /// erases `view`.
    public init<V>(_ view: V) where V :
View

    public init<V>(erasing view: V) where
V : View

    /// The type of view representing the
body of this view.
    ///
    /// When you create a custom view,
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Body = Never
}

/// A composite `Transition` that uses a
different transition for
/// insertion versus removal.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@MainActor @preconcurrency public struct
AsymmetricTransition<Insertion,
Removal> : Transition where Insertion :
Transition, Removal : Transition {

    /// The `Transition` defining the
insertion phase of `self`.
```

```
    @MainActor @preconcurrency public var  
insertion: Insertion
```

```
        /// The `Transition` defining the  
removal phase of `self`.
```

```
    @MainActor @preconcurrency public var  
removal: Removal
```

```
        /// Creates a composite `Transition`  
that uses a different transition for  
        /// insertion versus removal.
```

```
    @MainActor @preconcurrency public  
init(insertion: Insertion, removal:  
Removal)
```

```
        /// Gets the current body of the  
caller.
```

```
        ///  
        /// `content` is a proxy for the view  
that will have the modifier  
        /// represented by `Self` applied to  
it.
```

```
    @MainActor @preconcurrency public  
func body(content:  
AsymmetricTransition<Insertion,  
Removal>.Content, phase: TransitionPhase)  
-> some View
```

```
        /// Returns the properties this  
transition type has.
```

```
        ///  
        /// Defaults to
```

```
`TransitionProperties()`.  
    @MainActor @preconcurrency public  
static var properties:  
    TransitionProperties { get }  
  
        /// The type of view representing the  
body.  
        @available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
        public typealias Body = some View  
    }  
  
    /// The horizontal or vertical dimension  
in a 2D coordinate system.  
    @available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
    @frozen public enum Axis : Int8,  
CaseIterable {  
  
        /// The horizontal dimension.  
        case horizontal  
  
        /// The vertical dimension.  
        case vertical  
  
        /// An efficient set of axes.  
        @frozen public struct Set : OptionSet  
{  
  
            /// The element type of the  
option set.  
            ///  
            /// To inherit all the default
```

```
implementations from the `OptionSet` protocol,
```

```
    /// the `Element` type must be `Self`, the default.  
    public typealias Element = Axis.Set
```

```
        /// The corresponding value of the raw type.
```

```
        ///  
        /// A new instance initialized with `rawValue` will be equivalent to this
```

```
        /// instance. For example:  
        ///  
        ///     enum PaperSize: String {  
        ///         case A4, A5, Letter,  
Legal  
        ///     }  
        ///
```

```
        ///     let selectedSize =  
PaperSize.Letter  
        ///
```

```
print(selectedSize.rawValue)  
        /// // Prints "Letter"  
        ///
```

```
        ///     print(selectedSize ==  
PaperSize(rawValue:  
selectedSize.rawValue)!)  
        /// // Prints "true"
```

```
public let rawValue: Int8
```

```
        /// Creates a new option set from
```

the given raw value.

```
    /**
     * This initializer always succeeds, even if the value passed as `rawValue` exceeds the static properties declared as part of the option set. This example creates an instance of `ShippingOptions` with a raw value beyond the highest element, with a bit mask that effectively contains all the declared static members.
     */
    let extraOptions = ShippingOptions(rawValue: 255)
    print(extraOptions.isStrictSuperset(of: .all))
    // Prints "true"
    /**
     * - Parameter rawValue: The raw value of the option set to create. Each bit of `rawValue` potentially represents an element of the option set, though raw values may include bits that are not defined as distinct values of the `OptionSet` type.
     */
public init(rawValue: Int8)
```

```
        public static let horizontal:  
Axis.Set  
  
        public static let vertical:  
Axis.Set  
  
        /// The type of the elements of  
an array literal.  
        @available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
        public typealias  
ArrayLiteralElement = Axis.Set.Element  
  
        /// The raw type that can be used  
to represent all values of the conforming  
        /// type.  
        ///  
        /// Every distinct value of the  
conforming type has a corresponding  
unique  
        /// value of the `RawValue` type,  
but there may be values of the `RawValue`  
        /// type that don't have a  
corresponding value of the conforming  
type.  
        @available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
        public typealias RawValue = Int8  
    }  
  
    /// Creates a new instance with the  
specified raw value.
```

```
///  
/// If there is no value of the type  
that corresponds with the specified raw  
/// value, this initializer returns  
'nil'. For example:  
///  
///     enum PaperSize: String {  
///         case A4, A5, Letter,  
Legal  
///     }  
///  
///     print(PaperSize(rawValue:  
"Legal"))  
///     // Prints  
"Optional("PaperSize.Legal")"  
///  
///     print(PaperSize(rawValue:  
"Tabloid"))  
///     // Prints "nil"  
///  
/// - Parameter rawValue: The raw  
value to use for the new instance.  
public init?(rawValue: Int8)  
  
    /// A type that can represent a  
collection of all values of this type.  
@available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
public typealias AllCases = [Axis]  
  
    /// The raw type that can be used to  
represent all values of the conforming  
/// type.
```

```
///  
/// Every distinct value of the  
conforming type has a corresponding  
unique  
    /// value of the `RawValue` type, but  
there may be values of the `RawValue`  
    /// type that don't have a  
corresponding value of the conforming  
type.  
@available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
public typealias RawValue = Int8  
  
    /// A collection of all values of  
this type.  
nonisolated public static var  
allCases: [Axis] { get }  
  
    /// The corresponding value of the  
raw type.  
    ///  
    /// A new instance initialized with  
`rawValue` will be equivalent to this  
    /// instance. For example:  
    ///  
    ///     enum PaperSize: String {  
    ///         case A4, A5, Letter,  
Legal  
    ///     }  
    ///  
    ///     let selectedSize =  
PaperSize.Letter  
    ///     print(selectedSize.rawValue)
```

```
    ///      // Prints "Letter"
    ///
    ///      print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    ///      // Prints "true"
public var rawValue: Int8 { get }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Axis : CustomStringConvertible
{

    /// A textual representation of this
instance.
    ///
    /// Calling this property directly is
discouraged. Instead, convert an
    /// instance of any type to a string
by using the `String(describing:)`
    /// initializer. This initializer
works with any type, and uses the custom
    /// `description` property for types
that conform to
    /// `CustomStringConvertible`:
    ///
    ///     struct Point:
CustomStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var description: String {
    ///             return "(\(x), \(y))"

```

```
    /**
     *      }
     */
    /**
     *      let p = Point(x: 21, y: 30)
     *      let s = String(describing: p)
     *      print(s)
     *      // Prints "(21, 30)"
     */
    /**
     *      The conversion of `p` to a string
     *      in the assignment to `s` uses the
     *      `Point` type's `description`
     *      property.
     */
    public var description: String {
get }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Axis : Equatable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Axis : Hashable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Axis : RawRepresentable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
```

```
extension Axis : Sendable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Axis : BitwiseCopyable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Axis.Set : Sendable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Axis.Set : BitwiseCopyable {  
}  
  
/// The prominence of backgrounds  
underneath other views.  
///  
/// Background prominence should  
influence foreground styling to maintain  
/// sufficient contrast against the  
background. For example, selected rows in  
/// a `List` and `Table` can have  
increased prominence backgrounds with  
/// accent color fills when focused; the  
foreground content above the background  
/// should be adjusted to reflect that  
level of prominence.  
///  
/// This can be read and written for
```

```
views with the
///
`EnvironmentValues.backgroundProminence`
property.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct BackgroundProminence : Hashable, Sendable {

    /// The standard prominence of a
    background
    ///
    /// This is the default level of
    prominence and doesn't require any
    /// adjustment to achieve
    satisfactory contrast with the
    background.
    public static let standard:
BackgroundProminence

    /// A more prominent background that
    likely requires some changes to the
    /// views above it.
    ///
    /// This is the level of prominence
    for more highly saturated and full
    /// color backgrounds, such as
    focused/emphasized selected list rows.
    /// Typically foreground content
    should take on monochrome styling to
    /// have greater contrast against the
    background.
    public static let increased:
```

## BackgroundProminence

```
    /// 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:  
BackgroundProminence, b:  
BackgroundProminence) -> 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 }  
}
```

```
/// The background style in the current  
context.
```

```
///
```

```
/// You can also use
``ShapeStyle/background`` to construct
this style.
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
@frozen public struct BackgroundStyle : ShapeStyle {
    /// Creates a background style
    instance.
    @inlinable public init()
        /// The type of shape style this will
        resolve to.
        ///
        /// When you create a custom shape
        style, Swift infers this type
        /// from your implementation of the
        required `resolve` function.
        @available(iOS 17.0, tvOS 17.0,
        watchOS 10.0, macOS 14.0, *)
        public typealias Resolved = Never
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension BackgroundStyle : BitwiseCopyable {
}

/// A property wrapper type that supports
creating bindings to the mutable
/// properties of observable objects.
```

```
///  
/// Use this property wrapper to create  
bindings to mutable properties of a  
/// data model object that conforms to  
the  
///  
<doc://com.apple.documentation/documentation/Observation/Observable>  
/// protocol. For example, the following  
code wraps the `book` input with  
/// `@Bindable`. Then it uses a  
``TextField`` to change the `title`  
property of  
/// a book, and a ``Toggle`` to change  
the `isAvailable` property, using the  
/// `$` syntax to pass a binding for each  
property to those controls.  
///  
/// @Observable  
class Book: Identifiable {  
    var title = "Sample Book  
Title"  
    var isAvailable = true  
}  
  
struct BookEditView: View {  
    @Bindable var book: Book  
    @Environment(\.dismiss)  
private var dismiss  
    var body: some View {  
        Form {  
            TextField("Title",
```

```
text: $book.title)
///
///           Toggle("Book is
available", isOn: $book.isAvailable)
///
///           Button("Close") {
///
///               dismiss()
///
///           }
///
///       }
///
///   }
///
/// You can use the `Bindable` property
wrapper on properties and variables to
/// an
<doc://com.apple.documentation/documentation/Observation/Observable>
/// object. This includes global
variables, properties that exists outside
of
/// SwiftUI types, or even local
variables. For example, you can create a
/// `@Bindable` variable within a view's
``View/body-swift.property``:
///
///     struct LibraryView: View {
///         @State private var books =
[Book(), Book(), Book()]
///
///         var body: some View {
///             List(books) { book in
///                 @Bindable var book =
book
```

```
    ///             TextField("Title",
text: $book.title)
    ///         }
    ///     }
    /// }
    ///
/// The `@Bindable` variable `book`
provides a binding that connects
/// ``TextField`` to the `title` property
of a book so that a person can make
/// changes directly to the model data.
///
/// Use this same approach when you need
a binding to a property of an
/// observable object stored in a view's
environment. For example, the
/// following code uses the
``Environment`` property wrapper to
retrieve an
/// instance of the observable type
`Book`. Then the code creates a
`@Bindable`
/// variable `book` and passes a binding
for the `title` property to a
/// ``TextField`` using the `$` syntax.
///
///     struct TitleEditView: View {
///         @Environment(Book.self)
private var book
///
///         var body: some View {
///             @Bindable var book = book
///             TextField("Title", text:
```

```
$book.title)
    }
}
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@dynamicMemberLookup @propertyWrapper
public struct Bindable<Value> {

    /// The wrapped object.
    public var wrappedValue: Value

    /// The bindable wrapper for the
    object that creates bindings to its
    /// properties using dynamic member
    lookup.
    public var projectedValue:
Bindale<Value> { get }
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Bindable where Value :
AnyObject {

    /// Returns a binding to the value of
    a given key path.
    public
subscript<Subject>(dynamicMember keyPath:
ReferenceWritableKeyPath<Value, Subject>)
-> Binding<Subject> { get }
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Bindable where Value :  
AnyObject, Value : Observable {  
  
    /// Creates a bindable object from an  
observable object.  
    ///  
    /// You should not call this  
initializer directly. Instead, declare a  
    /// property with the `@Bindable`  
attribute, and provide an initial value.  
    public init(wrappedValue: Value)  
  
        /// Creates a bindable object from an  
observable object.  
        ///  
        /// This initializer is equivalent to  
``init(wrappedValue:)``, but is more  
        /// succinct when creating  
bindable objects nested within other  
        /// expressions. For example, you can  
use the initializer to create a  
        /// bindable object inline with code  
that declares a view that takes a  
        /// binding as a parameter:  
        ///  
        ///     struct TitleEditView: View {  
        ///         @Environment(Book.self)  
private var book  
        ///  
        ///             var body: some View {  
        ///                 TextField("Title",
```

```
text: Bindable(book).title)
    /**
     * }
     */
    /**
     * public init(_ wrappedValue: Value)

        /// Creates a bindable from the value
        of another bindable.
        public init(projectedValue:
Bindable<Value>)
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Bindable : Identifiable where
Value : Identifiable {

    /// The stable identity of the entity
    associated with this instance.
    public var id: Value.ID { get }

    /// A type representing the stable
    identity of the entity associated with
    /// an instance.
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias ID = Value.ID
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Bindable : Sendable where Value
: Sendable {
```

```
}
```

```
/// A property wrapper type that can read  
and write a value owned by a source of  
/// truth.
```

```
///
```

```
/// Use a binding to create a two-way  
connection between a property that stores  
/// data, and a view that displays and  
changes the data. A binding connects a  
/// property to a source of truth stored  
elsewhere, instead of storing data  
/// directly. For example, a button that  
toggles between play and pause can  
/// create a binding to a property of its  
parent view using the `Binding`  
/// property wrapper.
```

```
///
```

```
///     struct PlayButton: View {  
///         @Binding var isPlaying: Bool  
///  
///         var body: some View {  
///             Button(isPlaying ?  
"Pause" : "Play") {  
///                 isPlaying.toggle()  
///             }  
///         }  
///     }
```

```
/// The parent view declares a property  
to hold the playing state, using the  
/// ``State`` property wrapper to  
indicate that this property is the
```

```
value's
/// source of truth.
///
///     struct PlayerView: View {
///         var episode: Episode
///         @State private var.isPlaying:
Bool = false
///
///             var body: some View {
///                 VStack {
///                     Text(episode.title)
///                         .foregroundStyle(
///                             isPlaying ? .primary : .secondary)
///                     PlayButton(isPlaying:
$.isPlaying) // Pass a binding.
///                         }
///                     }
///                 }
///
/// When `PlayerView` initializes
`PlayButton`, it passes a binding of its
state
/// property into the button's binding
property. Applying the `$` prefix to a
/// property wrapped value returns its
``State/projectedValue``, which for a
/// state property wrapper returns a
binding to the value.
///
/// Whenever the user taps the
`PlayButton`, the `PlayerView` updates
its
/// `.isPlaying` state.
```

```
///  
/// A binding conforms to ``Sendable``  
only if its wrapped value type also  
/// conforms to ``Sendable``. It is  
always safe to pass a sendable binding  
/// between different concurrency  
domains. However, reading from or writing  
/// to a binding's wrapped value from a  
different concurrency domain may or  
/// may not be safe, depending on how the  
binding was created. SwiftUI will  
/// issue a warning at runtime if it  
detects a binding being used in a way  
/// that may compromise data safety.  
///  
/// > Note: To create bindings to  
properties of a type that conforms to the  
///  
 <doc://com.apple.documentation/documentat ion/Observation/Observable>  
/// protocol, use the ``Bindable``  
property wrapper. For more information,  
/// see <Migrating from the observable-object-protocol-to-the-observable-macro>.  
///  
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)  
@frozen @propertyWrapper  
@dynamicMemberLookup public struct Binding<Value> {  
  
    /// The binding's transaction.
```

```
///  
/// The transaction captures the  
information needed to update the view  
when  
/// the binding value changes.  
public var transaction: Transaction  
  
/// Creates a binding with closures  
that read and write the binding value.  
///  
/// A binding conforms to Sendable  
only if its wrapped value type also  
/// conforms to Sendable. It is  
always safe to pass a sendable binding  
/// between different concurrency  
domains. However, reading from or writing  
/// to a binding's wrapped value from  
a different concurrency domain may or  
/// may not be safe, depending on how  
the binding was created. SwiftUI will  
/// issue a warning at runtime if it  
detects a binding being used in a way  
/// that may compromise data safety.  
///  
/// For a "computed" binding created  
using get and set closure parameters,  
/// the safety of accessing its  
wrapped value from a different  
concurrency  
/// domain depends on whether those  
closure arguments are isolated to  
/// a specific actor. For example, a  
computed binding with closure arguments
```

```
    /// that are known (or inferred) to
    // be isolated to the main actor must only
    /// ever access its wrapped value on
    // the main actor as well, even if the
    /// binding is also sendable.
    ///
    /// - Parameters:
    ///   - get: A closure that retrieves
    // the binding value. The closure has no
    ///     parameters, and returns a
    // value.
    ///   - set: A closure that sets the
    // binding value. The closure has the
    ///     following parameter:
    ///       - newValue: The new value
    // of the binding value.
    @preconcurrency public init(get:
        @escaping @isolated(any) @Sendable () ->
        Value, set: @escaping @isolated(any)
        @Sendable (Value) -> Void)

    /// Creates a binding with a closure
    // that reads from the binding value, and
    /// a closure that applies a
    transaction when writing to the binding
    value.
    ///
    /// A binding conforms to Sendable
    only if its wrapped value type also
    /// conforms to Sendable. It is
    always safe to pass a sendable binding
    /// between different concurrency
    domains. However, reading from or writing
```

```
    /// to a binding's wrapped value from  
    a different concurrency domain may or  
    /// may not be safe, depending on how  
    the binding was created. SwiftUI will  
    /// issue a warning at runtime if it  
    detects a binding being used in a way  
    /// that may compromise data safety.  
    ///  
    /// For a "computed" binding created  
    using get and set closure parameters,  
    /// the safety of accessing its  
    wrapped value from a different  
    concurrency  
    /// domain depends on whether those  
    closure arguments are isolated to  
    /// a specific actor. For example, a  
    computed binding with closure arguments  
    /// that are known (or inferred) to  
    be isolated to the main actor must only  
    /// ever access its wrapped value on  
    the main actor as well, even if the  
    /// binding is also sendable.  
    ///  
    /// - Parameters:  
    ///   - get: A closure to retrieve  
    the binding value. The closure has no  
    ///   parameters, and returns a  
    value.  
    ///   - set: A closure to set the  
    binding value. The closure has the  
    ///   following parameters:  
    ///     - newValue: The new value  
    of the binding value.
```

```
    /// - transaction: The
transaction to apply when setting a new
value.
    @preconcurrency public init(get:
@escaping @isolated(any) @Sendable () ->
Value, set: @escaping @isolated(any)
@Sendable (Value, Transaction) -> Void)

    /// Creates a binding with an
immutable value.
    ///
    /// Use this method to create a
binding to a value that cannot change.
    /// This can be useful when using a
``PreviewProvider`` to see how a view
    /// represents different values.
    ///
    /// // Example of binding to an
immutable value.
    /// PlayButton(isPlaying:
Binding.constant(true))
    ///
    /// - Parameter value: An immutable
value.
    public static func constant(_ value:
Value) -> Binding<Value>

    /// The underlying value referenced
by the binding variable.
    ///
    /// This property provides primary
access to the value's data. However, you
    /// don't access `wrappedValue`
```

directly. Instead, you use the property  
``// variable created with the  
``Binding`` attribute.`` In the  
``// following code example, the  
binding variable `.isPlaying` returns the  
// value of `wrappedValue`:``  
``///`  
``/// struct PlayButton: View {`  
``/// @Binding var isPlaying:`  
`Bool`  
``///`  
``/// var body: some View {`  
``/// Button(isPlaying ?`  
``"Pause" : "Play") {`  
``///`  
``isPlaying.toggle()`  
``/// }`  
``/// }`  
``///`  
``/// When a mutable binding value`  
``changes, the new value is immediately`  
``/// available. However, updates to a`  
``view displaying the value happens`  
``/// asynchronously, so the view may`  
``not show the change immediately.`  
`public var wrappedValue: Value { get`  
``nonmutating set }`  
  
``/// A projection of the binding value`  
``that returns a binding.`  
``///`  
``/// Use the projected value to pass a`

binding value down a view hierarchy.

/// To get the `projectedValue`,  
prefix the property variable with `\$`.  
For

```
    /// example, in the following code  
example `PlayerView` projects a binding  
    /// of the state property `isPlaying`  
to the `PlayButton` view using  
    /// `$isPlaying`.  
    ///  
    ///     struct PlayerView: View {  
    ///         var episode: Episode  
    ///         @State private var  
isPlaying: Bool = false  
    ///  
    ///         var body: some View {  
    ///             VStack {  
    ///  
Text(episode.title)  
    ///                     .foregroundSt  
yle(isPlaying ? .primary : .secondary)  
    ///  
PlayButton(isPlaying: $isPlaying)  
    ///                     }  
    ///                     }  
    ///                 }  
    ///  
    public var projectedValue:  
Binding<Value> { get }  
  
    /// Creates a binding from the value  
of another binding.  
    public init(projectedValue:
```

```
Binding<Value>)

    /// Returns a binding to the
    resulting value of a given key path.
    ///
    /// - Parameter keyPath: A key path
    to a specific resulting value.
    ///
    /// - Returns: A new binding.
    public
    subscript<Subject>(dynamicMember keyPath:
    WritableKeyPath<Value, Subject>) ->
    Binding<Subject> { get }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Binding : @unchecked Sendable
where Value : Sendable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension Binding : Identifiable where
Value : Identifiable {

    /// The stable identity of the entity
    associated with this instance,
    /// corresponding to the `id` of the
    binding's wrapped value.
    public var id: Value.ID { get }

    /// A type representing the stable
```

```
identity of the entity associated with
    /// an instance.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias ID = Value.ID
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension Binding : Sequence where
Value : MutableCollection {

    /// A type representing the
sequence's elements.
    public typealias Element =
Binding<Value.Element>

    /// A type that provides the
sequence's iteration interface and
    /// encapsulates its iteration state.
    public typealias Iterator =
IndexingIterator<Binding<Value>>

    /// A collection representing a
contiguous subrange of this collection's
    /// elements. The subsequence shares
indices with the original collection.
    ///
    /// The default subsequence type for
collections that don't define their own
    /// is `Slice`.
    public typealias SubSequence =
Slice<Binding<Value>>
```

```
}
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Binding : Collection where  
Value : MutableCollection {  
  
    /// A type that represents a position  
    // in the collection.  
    ///  
    /// Valid indices consist of the  
    position of every element and a  
    /// "past the end" position that's  
    not valid for use as a subscript  
    /// argument.  
    public typealias Index = Value.Index  
  
    /// A type that represents the  
    indices that are valid for subscripting  
    the  
    /// collection, in ascending order.  
    public typealias Indices =  
Value.Indices  
  
    /// The position of the first element  
    in a nonempty collection.  
    ///  
    /// If the collection is empty,  
    `startIndex` is equal to `endIndex`.  
    public var startIndex:  
Binding<Value>.Index { get }  
  
    /// The collection's "past the end"
```

```
position---that is, the position one  
    /// greater than the last valid  
subscript argument.
```

```
    ///  
    /// When you need a range that  
includes the last element of a  
collection, use
```

```
    /// the half-open range operator  
(`..<`) with `endIndex`. The `..<  
operator
```

```
    /// creates a range that doesn't  
include the upper bound, so it's always  
    /// safe to use with `endIndex`. For  
example:
```

```
    ///  
    ///     let numbers = [10, 20, 30,  
40, 50]  
    ///     if let index =  
numbers.firstIndex(of: 30) {  
        ///         print(numbers[index ..<  
numbers.endIndex])  
        ///     }  
        /// // Prints "[30, 40, 50]"  
        ///  
    /// If the collection is empty,  
`endIndex` is equal to `startIndex`.  
    public var endIndex:  
Binding<Value>.Index { get }
```

```
    /// The indices that are valid for  
subscripting the collection, in ascending  
    /// order.  
    ///
```

```
    /// A collection's `indices` property  
    can hold a strong reference to the  
    /// collection itself, causing the  
    collection to be nonuniquely referenced.  
    /// If you mutate the collection  
    while iterating over its indices, a  
    strong  
        /// reference can result in an  
    unexpected copy of the collection. To  
    avoid  
        /// the unexpected copy, use the  
    `index(after:)` method starting with  
        /// `startIndex` to produce indices  
    instead.  
        ///  
        ///     var c =  
MyFancyCollection([10, 20, 30, 40, 50])  
        ///     var i = c.startIndex  
        ///     while i != c.endIndex {  
        ///         c[i] /= 5  
        ///         i = c.index(after: i)  
        ///     }  
        ///     // c == MyFancyCollection([2,  
4, 6, 8, 10])  
    public var indices: Value.Indices {  
get }  
  
    /// Returns the position immediately  
    after the given index.  
    ///  
    /// The successor of an index must be  
    well defined. For an index `i` into a  
    /// collection `c`, calling
```

```
`c.index(after: i)` returns the same
index every
    /// time.
    ///
    /// - Parameter i: A valid index of
the collection. `i` must be less than
    /// `endIndex`.
    /// - Returns: The index value
immediately after `i`.
public func index(after i:
Binding<Value>.Index) ->
Binding<Value>.Index

    /// Replaces the given index with its
successor.
    ///
    /// - Parameter i: A valid index of
the collection. `i` must be less than
    /// `endIndex`.
public func formIndex(after i: inout
Binding<Value>.Index)

    /// Accesses the element at the
specified position.
    ///
    /// The following example accesses an
element of an array through its
    /// subscript to print its value:
    ///
    ///     var streets = ["Adams",
"Bryant", "Channing", "Douglas",
"Evarts"]
    ///     print(streets[1])
```

```
    ///      // Prints "Bryant"
    ///
    /// You can subscript a collection
    with any valid index other than the
        /// collection's end index. The end
    index refers to the position one past
        /// the last element of a collection,
    so it doesn't correspond with an
        /// element.
    ///
    /// - Parameter position: The
    position of the element to access.
`position`
    /// must be a valid index of the
collection that is not equal to the
    /// `endIndex` property.
    ///
    /// - Complexity: O(1)
public subscript(position:
Binding<Value>.Index) ->
Binding<Value>.Element { get }
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension Binding :
BidirectionalCollection where Value :
BidirectionalCollection, Value :
MutableCollection {

    /// Returns the position immediately
before the given index.
    ///
```

```
    /// - Parameter i: A valid index of  
the collection. `i` must be greater than  
    /// `startIndex`.  
    /// - Returns: The index value  
immediately before `i`.  
    public func index(before i:  
Binding<Value>.Index) ->  
Binding<Value>.Index  
  
    /// Replaces the given index with its  
predecessor.  
    ///  
    /// - Parameter i: A valid index of  
the collection. `i` must be greater than  
    /// `startIndex`.  
    public func formIndex(before i: inout  
Binding<Value>.Index)  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Binding :  
RandomAccessCollection where Value :  
MutableCollection, Value :  
RandomAccessCollection {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Binding {  
  
    /// Specifies a transaction for the  
binding.
```

```
///  
/// - Parameter transaction : An  
instance of a ``Transaction``.  
///  
/// - Returns: A new binding.  
public func transaction(_  
transaction: Transaction) ->  
Binding<Value>  
  
    /// Specifies an animation to perform  
when the binding value changes.  
    ///  
    /// - Parameter animation: An  
animation sequence performed when the  
binding  
    ///     value changes.  
    ///  
    /// - Returns: A new binding.  
    public func animation(_ animation:  
Animation? = .default) -> Binding<Value>  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Binding : DynamicProperty {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Binding {  
  
    /// Creates a binding by projecting  
the base value to an optional value.
```

```
///  
/// - Parameter base: A value to  
project to an optional value.  
public init<V>(_ base: Binding<V>)  
where Value == V?  
  
    /// Creates a binding by projecting  
the base value to an unwrapped value.  
    ///  
    /// - Parameter base: A value to  
project to an unwrapped value.  
    ///  
    /// - Returns: A new binding or `nil`  
when `base` is `nil`.  
public init?(_ base: Binding<Value?>)  
  
    /// Creates a binding by projecting  
the base value to a hashable value.  
    ///  
    /// - Parameters:  
    ///     - base: A `Hashable` value to  
project to an `AnyHashable` value.  
    public init<V>(_ base: Binding<V>)  
where Value == AnyHashable, V : Hashable  
}  
  
/// Modes for compositing a view with  
overlapping content.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
public enum BlendMode : Sendable {  
  
    case normal
```

case multiply  
case screen  
case overlay  
case darken  
case lighten  
case colorDodge  
case colorBurn  
case softLight  
case hardLight  
case difference  
case exclusion  
case hue  
case saturation  
case color  
case luminosity  
case sourceAtop

```
    case destinationOver  
  
    case destinationOut  
  
    case plusDarker  
  
    case plusLighter  
  
    /// 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: BlendMode,  
b: BlendMode) -> Bool  
  
    /// 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)
```

```
    /// 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 }
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
extension BlendMode : Equatable {  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
extension BlendMode : Hashable {  
}
```

```
/// A transition that animates the  
insertion or removal of a view by  
/// combining blurring and scaling  
effects.
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)
```

```
@MainActor @preconcurrency public struct  
BlurReplaceTransition : Transition {
```

```
    /// Configuration properties for a  
transition.
```

```
    public struct Configuration :  
Equatable {
```

```
        /// A configuration that requests  
a transition that scales the
```

```
        /// view down while removing it  
and up while inserting it.
```

```
        public static let downUp:  
BlurReplaceTransition.Configuration
```

```
        /// A configuration that requests
a transition that scales the
        /// view up while both removing
and inserting it.
    public static let upUp:
BlurReplaceTransition.Configuration

        /// 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:
BlurReplaceTransition.Configuration, b:
BlurReplaceTransition.Configuration) ->
Bool
}

/// The transition configuration.
@MainActor @preconcurrency public var
configuration:
BlurReplaceTransition.Configuration

        /// Creates a new transition.
        ///
        /// - Parameter configuration: the
```

```
transition configuration.
```

```
    @MainActor @preconcurrency public  
init(configuration:  
BlurReplaceTransition.Configuration)
```

```
        /// Gets the current body of the  
caller.
```

```
        ///  
        /// `content` is a proxy for the view  
that will have the modifier  
        /// represented by `Self` applied to  
it.
```

```
    @MainActor @preconcurrency public  
func body(content:  
BlurReplaceTransition.Content, phase:  
TransitionPhase) -> some View
```

```
        /// The type of view representing the  
body.
```

```
        @available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
        public typealias Body = some View  
    }
```

```
    /// A view type that supports immediate  
mode drawing.
```

```
    ///  
    /// Use a canvas to draw rich and dynamic  
2D graphics inside a SwiftUI view.
```

```
    /// The canvas passes a  
``GraphicsContext`` to the closure that  
you use
```

```
/// to perform immediate mode drawing
operations. The canvas also passes a
///
<doc://com.apple.documentation/documentation/CoreFoundation/CGSize> value
/// that you can use to customize what
you draw. For example, you can use the
/// context's
``GraphicsContext/stroke(_:with:lineWidth
:)`` command to draw
/// a ``Path`` instance:
///
///     Canvas { context, size in
///         context.stroke(
///             Path(ellipseIn:
CGRect(origin: .zero, size: size)),
///             with: .color(.green),
///             lineWidth: 4)
///     }
///     .frame(width: 300, height: 200)
///     .border(Color.blue)
///

/// The example above draws the outline
of an ellipse that exactly inscribes
/// a canvas with a blue border:
///
/// ! [A screenshot of a canvas view that
shows the green outline of an
/// ellipse inside a blue rectangle.]
(Canvas-1)
///
/// In addition to outlined and filled
paths, you can draw images, text, and
```

```
/// complete SwiftUI views. To draw
views, use the
///
``init(opaque:colorMode:rendersAsynchronously:renderer:symbols:)`` method
/// to supply views that you can
reference from inside the renderer. You
can
/// also add masks, apply filters,
perform transforms, control blending, and
/// more. For information about how to
draw, see ``GraphicsContext``.
///
/// A canvas doesn't offer interactivity
or accessibility for
/// individual elements, including for
views that you pass in as symbols.
/// However, it might provide better
performance for a complex drawing that
/// involves dynamic data. Use a canvas
to improve performance for a drawing
/// that doesn't primarily involve text
or require interactive elements.
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
public struct Canvas<Symbols> where
Symbols : View {
    /// A view that provides child views
that you can use in the drawing
    /// callback.
    ///
    /// Uniquely tag each child view
```

```
using the ``View/tag(_:)`` modifier,  
    /// so that you can find them from  
within your renderer using the  
    ///  
``GraphicsContext/resolveSymbol(id:)``  
method.  
public var symbols: Symbols  
  
    /// The drawing callback that you use  
to draw into the canvas.  
    ///  
    /// - Parameters:  
    ///   - context: The graphics context  
to draw into.  
    ///   - size: The current size of the  
view.  
public var renderer: (inout  
GraphicsContext, CGSize) -> Void  
  
    /// A Boolean that indicates whether  
the canvas is fully opaque.  
    ///  
    /// You might be able to improve  
performance by setting this value to  
    /// `true`, making the canvas is  
fully opaque. However, in that case,  
    /// the result of drawing a non-  
opaque image into the canvas is  
undefined.  
public var isOpaque: Bool  
  
    /// The working color space and  
storage format of the canvas.
```

```
public var colorMode:  
ColorRenderingMode  
  
    /// A Boolean that indicates whether  
the canvas can present its contents  
    /// to its parent view  
asynchronously.  
public var rendersAsynchronously:  
Bool  
  
    /// Creates and configures a canvas  
that you supply with renderable  
    /// child views.  
    ///  
    /// This initializer behaves like the  
    ///  
``init(opaque:colorMode:rendersAsynchronously:renderer:)`` initializer,  
    /// except that you also provide a  
collection of SwiftUI views for the  
    /// renderer to use as drawing  
elements.  
    ///  
    /// SwiftUI stores a rendered version  
of each child view that you specify  
    /// in the `symbols` view builder and  
makes these available to the canvas.  
    /// Tag each child view so that you  
can retrieve it from within the  
    /// renderer using the  
``GraphicsContext/resolveSymbol(id:)``  
method.  
    /// For example, you can create a
```

```
scatter plot using a passed-in child view
    /// as the mark for each data point:
    ///
    /// struct ScatterPlotView<Mark:
View>: View {
    ///
    let rects: [CGRect]
    let mark: Mark
    ///
    enum SymbolID: Int {
        case mark
    }
    ///
    var body: some View {
        Canvas { context,
size in
        ///
        if let mark =
context.resolveSymbol(id: SymbolID.mark)
{
            ///
            for rect in
rects {
                ///
                context.draw(mark, in: rect)
                ///
            }
            ///
        } symbols: {
            ///
            mark.tag(SymbolID.mark)
            ///
        }
        ///
        .frame(width: 300,
height: 200)
        ///
        .border(Color.blue)
        ///
    }
}
```

```
///  
/// You can use any SwiftUI view for  
the `mark` input:  
///  
///     ScatterPlotView(rects: rects,  
mark: Image(systemName: "circle"))  
///  
/// If the `rects` input contains 50  
randomly arranged  
///  
<doc://com.apple.documentation/documentation/CoreFoundation/CGRect>  
/// instances, SwiftUI draws a plot  
like this:  
///  
/// ![A screenshot of a scatter plot  
inside a blue rectangle, containing  
/// about fifty small circles  
scattered randomly throughout.] (Canvas-  
init-1)  
///  
/// The symbol inputs, like all other  
elements that you draw to the  
/// canvas, lack individual  
accessibility and interactivity, even if  
the  
/// original SwiftUI view has these  
attributes. However, you can add  
/// accessibility and interactivity  
modifers to the canvas as a whole.  
///  
/// - Parameters:  
///   - opaque: A Boolean that
```

indicates whether the canvas is fully opaque. You might be able to improve performance by setting this value to `true`, but then drawing a non-opaque image into the context produces undefined results. The default is `false`.

/// - colorMode: A working color space and storage format of the canvas.

/// The default is ``ColorRenderingMode/nonLinear``.

/// - rendersAsynchronously: A Boolean that indicates whether the canvas can present its contents to its parent view asynchronously. The default is `false`.

/// - renderer: A closure in which you conduct immediate mode drawing.

/// The closure takes two inputs: a context that you use to issue drawing commands and a size --- representing the current size of the canvas --- that you can use to customize the content.

/// The canvas calls the renderer any time it needs to redraw the content.

/// - symbols: A ``ViewBuilder`` that you use to supply SwiftUI views to the canvas for use during drawing. Uniquely tag each view using the ``View/tag(\_:)`` modifier, so that you can find them from

```
    ///      within your renderer using
the
``GraphicsContext/resolveSymbol(id)``
    ///      method.
    public init(opaque: Bool = false,
colorMode: ColorRenderingMode
= .nonLinear, rendersAsynchronously: Bool
= false, renderer: @escaping (inout
GraphicsContext, CGSize) -> Void,
@ViewBuilder symbols: () -> Symbols)

    /// The type of view representing the
body of this view.
    ///
    /// When you create a custom view,
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias Body = Never
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension Canvas where Symbols ==
EmptyView {

    /// Creates and configures a canvas.
    ///
    /// Use this initializer to create a
new canvas that you can draw into.
    /// For example, you can draw a path:
```

```
///
///      Canvas { context, size in
///          context.stroke(
///              Path(ellipseIn:
///                  CGRect(origin: .zero, size: size)),
///                  with: .color(.green),
///                  lineWidth: 4)
///          }
///          .frame(width: 300, height:
200)
///          .border(Color.blue)
///
/// The example above draws the
outline of an ellipse that exactly
inscribes
/// a canvas with a blue border:
///
/// ! [A screenshot of a canvas view
that shows the green outline of an
/// ellipse inside a blue rectangle.]  

(Canvas-1)
///
/// For information about using a
context to draw into a canvas, see
/// ``GraphicsContext``. If you want
to provide SwiftUI views for the
/// renderer to use as drawing
elements, use
///
``init(opaque:colorMode:rendersAsynchronously:renderer:symbols:)``
/// instead.
///
```

```
    /// - Parameters:  
    /// - opaque: A Boolean that  
    indicates whether the canvas is fully  
    /// opaque. You might be able to  
    improve performance by setting this  
    /// value to `true`, but then  
    drawing a non-opaque image into the  
    /// context produces undefined  
    results. The default is `false`.  
    /// - colorMode: A working color  
    space and storage format of the canvas.  
    /// The default is  
    ``ColorRenderingMode/nonLinear``.  
    /// - rendersAsynchronously: A  
    Boolean that indicates whether the canvas  
    /// can present its contents to  
    its parent view asynchronously. The  
    /// default is `false`.  
    /// - renderer: A closure in which  
    you conduct immediate mode drawing.  
    /// The closure takes two inputs:  
    a context that you use to issue  
    /// drawing commands and a size  
    --- representing the current  
    /// size of the canvas --- that  
    you can use to customize the content.  
    /// The canvas calls the renderer  
    any time it needs to redraw the  
    /// content.  
    public init(opaque: Bool = false,  
    colorMode: ColorRenderingMode  
    = .nonLinear, rendersAsynchronously: Bool  
    = false, renderer: @escaping (inout
```

```
    GraphicsContext, CGSize) -> Void)
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension Canvas : View {

/// A capsule shape aligned inside the
frame of the view containing it.
///
/// A capsule shape is equivalent to a
rounded rectangle where the corner radius
/// is chosen as half the length of the
rectangle's smallest edge.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct Capsule : Shape {

    public var style: RoundedCornerStyle

        /// Creates a new capsule shape.
        ///
        /// - Parameters:
        ///     - style: the style of corners
        ///     drawn by the shape.
        @inlinable public init(style:
RoundedCornerStyle = .continuous)

        /// Describes this shape as a path
        /// within a rectangular frame of reference.
        ///
        /// - Parameter rect: The frame of
```

reference for describing this shape.

```
///  
/// - Returns: A path that describes  
this shape.  
nonisolated public func path(in r:  
CGRect) -> Path
```

```
/// Returns the behavior this shape  
should use for different layout  
/// directions.  
///  
/// If the layoutDirectionBehavior  
for a Shape is one that mirrors, the  
/// shape's path will be mirrored  
horizontally when in the specified layout  
/// direction. When mirrored, the  
individual points of the path will be  
/// transformed.  
///  
/// Defaults to `mirrors` when  
deploying on iOS 17.0, macOS 14.0,  
/// tvOS 17.0, watchOS 10.0 and  
later, and to `fixed` if not.  
/// To mirror a path when deploying  
to earlier releases, either use  
///  
`View.flipsForRightToLeftLayoutDirection`  
for a filled or stroked  
/// shape or conditionally mirror the  
points in the path of the shape.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
nonisolated public var
```

```
layoutDirectionBehavior:  
LayoutDirectionBehavior { get }  
  
    /// The type defining the data to  
animate.  
    @available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
    public typealias AnimatableData =  
EmptyAnimatableData  
  
    /// The type of view representing the  
body of this view.  
    ///  
    /// When you create a custom view,  
Swift infers this type from your  
    /// implementation of the required  
``View/body-swift.property`` property.  
    @available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
    public typealias Body  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Capsule : InsettableShape {  
  
    /// Returns `self` inset by `amount`.  
    @inlinable nonisolated public func  
inset(by amount: CGFloat) -> some  
InsettableShape  
  
    /// The type of the inset shape.
```

```
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias InsetShape = some
InsettableShape
}

/// A circle centered on the frame of the
view containing it.
///
/// The circle's radius equals half the
length of the frame rectangle's smallest
/// edge.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct Circle : Shape {

    /// Describes this shape as a path
within a rectangular frame of reference.
    ///
    /// - Parameter rect: The frame of
reference for describing this shape.
    ///
    /// - Returns: A path that describes
this shape.
    nonisolated public func path(in rect:
CGRect) -> Path

    /// Creates a new circle shape.
    @inlinable public init()

    /// Returns the behavior this shape
should use for different layout
    /// directions.
```

```
///  
/// If the layoutDirectionBehavior  
for a Shape is one that mirrors, the  
/// shape's path will be mirrored  
horizontally when in the specified layout  
/// direction. When mirrored, the  
individual points of the path will be  
/// transformed.  
///  
/// Defaults to `mirrors` when  
deploying on iOS 17.0, macOS 14.0,  
/// tvOS 17.0, watchOS 10.0 and  
later, and to `fixed` if not.  
/// To mirror a path when deploying  
to earlier releases, either use  
///  
`View.flipsForRightToLeftLayoutDirection`  
for a filled or stroked  
/// shape or conditionally mirror the  
points in the path of the shape.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
nonisolated public var  
layoutDirectionBehavior:  
LayoutDirectionBehavior { get }  
  
/// The type defining the data to  
animate.  
@available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
public typealias AnimatableData =  
EmptyAnimatableData
```

```
    /// The type of view representing the
body of this view.
    /**
     /// When you create a custom view,
Swift infers this type from your
     /// implementation of the required
``View/body-swift.property`` property.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Body
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension Circle {

    /// Returns the size of the view that
will render the shape, given
    /// a proposed size.
    /**
     /// Implement this method to tell the
container of the shape how
     /// much space the shape needs to
render itself, given a size
    /// proposal.
    /**
     /// See
``Layout/sizeThatFits(proposal:subviews:c
ache:)``
    /// for more details about how the
layout system chooses the size of
    /// views.
    /**
```

```
    /// - Parameters:  
    ///   - proposal: A size proposal for  
the container.  
    ///  
    /// - Returns: A size that indicates  
how much space the shape needs.  
    nonisolated public func  
sizeThatFits(_ proposal:  
ProposedViewSize) -> CGSize  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Circle : InsettableShape {  
  
    /// Returns `self` inset by `amount`.  
    @inlinable nonisolated public func  
inset(by amount: CGFloat) -> some  
InsettableShape  
  
    /// The type of the inset shape.  
    @available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
    public typealias InsetShape = some  
InsettableShape  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Circle : BitwiseCopyable {  
}
```

```
/// A representation of a color that  
adapts to a given context.  
///  
/// You can create a color in one of  
several ways:  
///  
/// * Load a color from an Asset Catalog:  
///     ````  
///         let aqua = Color("aqua") // Looks  
in your app's main bundle by default.  
///     ````  
/// * Specify component values, like red,  
green, and blue; hue,  
/// saturation, and brightness; or  
white level:  
///     ````  
///         let skyBlue = Color(red: 0.4627,  
green: 0.8392, blue: 1.0)  
///         let lemonYellow = Color(hue:  
0.1639, saturation: 1, brightness: 1)  
///         let steelGray = Color(white:  
0.4745)  
///     ````  
/// * Create a color instance from  
another color, like a  
///  
 <doc://com.apple.documentation/documentation/UIKit/UICOLOR> or an  
///  
 <doc://com.apple.documentation/documentation/AppKit/NSColor>:  
///     ````  
///         #if os(iOS)
```

```
///      let linkColor =
Color(uiColor: .link)
///      #elseif os(macOS)
///      let linkColor =
Color(nsColor: .linkColor)
///      #endif
///      ```
/// * Use one of a palette of predefined
colors, like ``ShapeStyle/black``,
/// ``ShapeStyle/green``, and
``ShapeStyle/purple``.
///
/// Some view modifiers can take a color
as an argument. For example,
/// ``View/foregroundStyle(_:)`` uses the
color you provide to set the
/// foreground color for view elements,
like text or
///
<doc://com.apple.documentation/design/hum
an-interface-guidelines/sf-symbols>:
///
///      Image(systemName: "leaf.fill")
///              .foregroundStyle(Color.green)
///
/// ! [A screenshot of a green leaf.]  

(Color-1)
///
/// Because SwiftUI treats colors as
``View`` instances, you can also
/// directly add them to a view
hierarchy. For example, you can layer
/// a rectangle beneath a sun image using
```

colors defined above:

```
///  
///     ZStack {  
///         skyBlue  
///         Image(systemName:  
"sun.max.fill")  
///             .foregroundStyle(lemonYel  
low)  
///     }  
///     .frame(width: 200, height: 100)  
///  
/// A color used as a view expands to  
fill all the space it's given,  
/// as defined by the frame of the  
enclosing ``ZStack`` in the above  
example:  
///  
/// ! [A screenshot of a yellow sun on a  
blue background.] (Color-2)  
///  
/// SwiftUI only resolves a color to a  
concrete value  
/// just before using it in a given  
environment.  
/// This enables a context-dependent  
appearance for  
/// system defined colors, or those that  
you load from an Asset Catalog.  
/// For example, a color can have  
distinct light and dark variants  
/// that the system chooses from at  
render time.  
@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
@frozen public struct Color : Hashable,
CustomStringConvertible, Sendable {

    /// Creates a color that represents
    the specified custom color.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public init<T>(_ color: T) where T :
    Hashable, T : ShapeStyle, T.Resolved ==
    Color.Resolved

    /// Evaluates this color to a
    resolved color given the current
    /// `context`.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public func resolve(in environment:
    EnvironmentValues) -> Color.Resolved

    /// A Core Graphics representation of
    the color, if available.
    ///
    /// You can get a
    ///
<doc://com.apple.documentation/documentation/CoreGraphics/CGColor>
    /// instance from a constant SwiftUI
    color. This includes colors you create
    /// from a Core Graphics color, from
    RGB or HSB components, or from constant
    /// UIKit and AppKit colors.
    ///
```

```
    /// For a dynamic color, like one you
    load from an Asset Catalog using
    /// ``init(_:bundle:)``, or one you
    create from a dynamic UIKit or AppKit
    /// color, this property is `nil`. To
    evaluate all types of colors, use the
    /// `resolve(in:)` method.
    @available(iOS, introduced: 14.0,
    deprecated: 100000.0, renamed:
    "resolve(in:)")
    @available(macOS, introduced: 11.0,
    deprecated: 100000.0, renamed:
    "resolve(in:)")
    @available(tvOS, introduced: 14.0,
    deprecated: 100000.0, renamed:
    "resolve(in:)")
    @available(watchOS, introduced: 7.0,
    deprecated: 100000.0, renamed:
    "resolve(in:)")
    @available(visionOS, introduced: 1.0,
    deprecated: 100000.0, renamed:
    "resolve(in:)")
    public var cgColor: CGColor? { get }

    /// Hashes the essential components
    of the color by feeding them into the
    /// given hash function.
    ///
    /// - Parameters:
    ///   - hasher: The hash function to
    use when combining the components of
    ///   the color.
    public func hash(into hasher: inout
```

Hasher)

```
    /// Indicates whether two colors are
    /// equal.
    ///
    /// - Parameters:
    ///   - lhs: The first color to
    ///     compare.
    ///   - rhs: The second color to
    ///     compare.
    /// - Returns: A Boolean that's set
    ///   to `true` if the two colors are equal.
    public static func == (lhs: Color,
    rhs: Color) -> Bool

    /// A textual representation of the
    /// color.
    ///
    /// Use this method to get a string
    /// that represents the color.
    /// The
<doc://com.apple.documentation/documentation/Swift/print(_:_separator:_terminator:)>
    /// function uses this property to
    /// get a string representing an instance:
    ///
    ///     print(Color.red)
    ///     // Prints "red"
    public var description: String {
get }

    /// 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 }
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Color : ShapeStyle {  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Color {
```

```
    /// A color that reflects the accent  
color of the system or app.
```

```
    ///
```

```
    /// The accent color is a broad theme  
color applied to
```

```
    /// views and controls. You can set  
it at the application level by specifying
```

```
    /// an accent color in your app's
```

```
asset catalog.

    /**
     * > Note: In macOS, SwiftUI applies
     * customization of the accent color
     * only if the user chooses
     * Multicolor under General > Accent color
     * in System Preferences.

    /**
     * The following code renders a
     ``Text`` view using the app's accent
     color:
    /**
     /**
      Text("Accent Color")
      .foregroundStyle(Color.ac
centColor)
    /**
        public static var accentColor: Color
    { get }
}

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

    /**
     * A concrete color value.

    /**
     * `Color.Resolved` is a set of RGBA
     values that represent a color that can
     * be shown. The values are in
     Linear sRGB color space, extended range.
This is
    /**
     * a low-level type, most colors are
     represented by the `Color` type.
```

```
///  
/// - SeeAlso: `Color`  
@frozen public struct Resolved :  
Hashable {  
  
    /// The amount of red in the  
color in the sRGB linear color space.  
    public var linearRed: Float  
  
    /// The amount of green in the  
color in the sRGB linear color space.  
    public var linearGreen: Float  
  
    /// The amount of blue in the  
color in the sRGB linear color space.  
    public var linearBlue: Float  
  
    /// The degree of opacity in the  
color, given in the range `0` to `1`.  
    ///  
    /// A value of `0` means 100%  
transparency, while a value of `1` means  
    /// 100% opacity.  
    public var opacity: Float  
  
    /// 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:
Color.Resolved, b: Color.Resolved) ->
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 }
}

    /// Creates a constant color with the
values specified by the resolved
    /// color.
    public init(_ resolved:
Color.Resolved)
{
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Color {  
  
    /// A profile that specifies how to  
    // interpret a color value for display.  
    public enum RGBColorSpace : Sendable  
{  
  
        /// The extended red, green, blue  
        // (sRGB) color space.  
        ///  
        /// For information about the  
        // sRGB colorimetry and nonlinear  
        /// transform function, see the  
        // IEC 61966-2-1 specification.  
        ///  
        /// Standard sRGB color spaces  
        // clamp the red, green, and blue  
        /// components of a color to a  
        // range of `0` to `1`, but SwiftUI colors  
        /// use an extended sRGB color  
        // space, so you can use component values  
        /// outside that range.  
        case sRGB  
  
        /// The extended sRGB color space  
        // with a linear transfer function.  
        ///  
        /// This color space has the same  
        // colorimetry as ``sRGB``, but uses  
        /// a linear transfer function.  
        ///
```

```
    /// Standard sRGB color spaces
    clamp the red, green, and blue
        /// components of a color to a
        range of `0` to `1`, but SwiftUI colors
        /// use an extended sRGB color
        space, so you can use component values
        /// outside that range.
    case sRGBLinear

        /// The Display P3 color space.
        ///
        /// This color space uses the
        Digital Cinema Initiatives – Protocol 3
            /// (DCI–P3) primary colors, a
        D65 white point, and the ``sRGB``
            /// transfer function.
    case displayP3

        /// 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:
Color.RGBColorSpace, b:
Color.RGBColorSpace) -> Bool
```

```
    /// 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)
```

```
    /// 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 }
}

/**
 * Creates a constant color from
 * red, green, and blue component values.
 *
 * This initializer creates a
 * constant color that doesn't change based
 * on context. For example, it
 * doesn't have distinct light and dark
 * appearances, unlike various
 * system-defined colors, or a color that
 * you load from an Asset Catalog
 * with ``init(_:bundle:)``.
 *
 * A standard sRGB color space
 * clamps each color component – `red`,
```

```
    /// `green`, and `blue` – to a range
of `0` to `1`, but SwiftUI colors
    /// use an extended sRGB color space,
so
        /// you can use component values
outside that range. This makes it
        /// possible to create colors using
the ``RGBColorSpace/sRGB`` or
        /// ``RGBColorSpace/sRGBLinear``
color space that make full use of the
wider
    /// gamut of a display that supports
``RGBColorSpace/displayP3``.

    ///
    /// - Parameters:
    ///   - colorSpace: The profile that
specifies how to interpret the color
    ///   for display. The default is
``RGBColorSpace/sRGB``.
    ///   - red: The amount of red in the
color.
    ///   - green: The amount of green in
the color.
    ///   - blue: The amount of blue in
the color.
    ///   - opacity: An optional degree
of opacity, given in the range `0` to
    ///   `1`. A value of `0` means
100% transparency, while a value of `1`
    ///   means 100% opacity. The
default is `1`.

public init(_ colorSpace:
Color.RGBColorSpace = .sRGB, red: Double,
```

```
green: Double, blue: Double, opacity:  
Double = 1)  
  
    /// Creates a constant grayscale  
color.  
    ///  
    /// This initializer creates a  
constant color that doesn't change based  
    /// on context. For example, it  
doesn't have distinct light and dark  
    /// appearances, unlike various  
system-defined colors, or a color that  
    /// you load from an Asset Catalog  
with ``init(_:bundle:)``.  
    ///  
    /// A standard sRGB color space  
clamps the `white` component  
    /// to a range of `0` to `1`, but  
SwiftUI colors  
    /// use an extended sRGB color space,  
so  
    /// you can use component values  
outside that range. This makes it  
    /// possible to create colors using  
the ``RGBColorSpace/sRGB`` or  
    /// ``RGBColorSpace/sRGBLinear``  
color space that make full use of the  
wider  
    /// gamut of a display that supports  
``RGBColorSpace/displayP3``.  
    ///  
    /// - Parameters:  
    ///     - colorSpace: The profile that
```

```
specifies how to interpret the color
    ///      for display. The default is
``RGBColorSpace/sRGB``.
    ///      - white: A value that indicates
how white
        ///      the color is, with higher
values closer to 100% white, and lower
        ///      values closer to 100% black.
        ///      - opacity: An optional degree
of opacity, given in the range `0` to
        ///      `1`. A value of `0` means
100% transparency, while a value of `1`
        ///      means 100% opacity. The
default is `1`.
public init(_ colorSpace:
Color.RGBColorSpace = .sRGB, white:
Double, opacity: Double = 1)

    /// Creates a constant color from
hue, saturation, and brightness values.
    ///
    /// This initializer creates a
constant color that doesn't change based
    /// on context. For example, it
doesn't have distinct light and dark
    /// appearances, unlike various
system-defined colors, or a color that
    /// you load from an Asset Catalog
with ``init(_:bundle:)``.
    ///
    /// - Parameters:
    ///      - hue: A value in the range `0`
to `1` that maps to an angle
```

```
    /// from 0° to 360° to represent  
a shade on the color wheel.
```

```
    /// - saturation: A value in the  
range `0` to `1` that indicates
```

```
    /// how strongly the hue affects  
the color. A value of `0` removes the
```

```
    /// effect of the hue, resulting  
in gray. As the value increases,
```

```
    /// the hue becomes more  
prominent.
```

```
    /// - brightness: A value in the  
range `0` to `1` that indicates
```

```
    /// how bright a color is. A  
value of `0` results in black, regardless
```

```
    /// of the other components. The  
color lightens as you increase this
```

```
    /// component.
```

```
    /// - opacity: An optional degree  
of opacity, given in the range `0` to
```

```
    /// `1`. A value of `0` means  
100% transparency, while a value of `1`
```

```
    /// means 100% opacity. The  
default is `1`.
```

```
    public init(hue: Double, saturation:  
Double, brightness: Double, opacity:  
Double = 1)  
}
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension Color {
```

```
    /// Returns the standard gradient for
```

```
the color `self`.  
    ///  
    /// For example, filling a rectangle  
with a gradient derived from  
    /// the standard blue color:  
    ///  
    ///  
    ///  
    Rectangle().fill(.blue.gradient)  
    ///  
    public var gradient: AnyGradient {  
get }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Color {  
  
    /// A context-dependent red color  
suitable for use in UI elements.  
    public static let red: Color  
  
    /// A context-dependent orange color  
suitable for use in UI elements.  
    public static let orange: Color  
  
    /// A context-dependent yellow color  
suitable for use in UI elements.  
    public static let yellow: Color  
  
    /// A context-dependent green color  
suitable for use in UI elements.  
    public static let green: Color
```

```
    /// A context-dependent mint color  
    suitable for use in UI elements.  
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    public static let mint: Color  
  
    /// A context-dependent teal color  
    suitable for use in UI elements.  
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    public static let teal: Color  
  
    /// A context-dependent cyan color  
    suitable for use in UI elements.  
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    public static let cyan: Color  
  
    /// A context-dependent blue color  
    suitable for use in UI elements.  
    public static let blue: Color  
  
    /// A context-dependent indigo color  
    suitable for use in UI elements.  
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    public static let indigo: Color  
  
    /// A context-dependent purple color  
    suitable for use in UI elements.  
    public static let purple: Color  
  
    /// A context-dependent pink color
```

```
suitable for use in UI elements.  
    public static let pink: Color  
  
        /// A context-dependent brown color  
        suitable for use in UI elements.  
        @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    public static let brown: Color  
  
        /// A white color suitable for use in  
        UI elements.  
    public static let white: Color  
  
        /// A context-dependent gray color  
        suitable for use in UI elements.  
    public static let gray: Color  
  
        /// A black color suitable for use in  
        UI elements.  
    public static let black: Color  
  
        /// A clear color suitable for use in  
        UI elements.  
    public static let clear: Color  
  
        /// The color to use for primary  
        content.  
    public static let primary: Color  
  
        /// The color to use for secondary  
        content.  
    public static let secondary: Color  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Color : View {  
  
    /// The type of view representing the  
    body of this view.  
    ///  
    /// When you create a custom view,  
    Swift infers this type from your  
    /// implementation of the required  
    ``View/body-swift.property`` property.  
    public typealias Body = Never  
}  
  
@available(iOS, introduced: 14.0,  
deprecated: 100000.0, message: "Use  
Color(cgColor:) when converting a  
CGColor, or create a standard Color  
directly")  
@available(macOS, introduced: 11.0,  
deprecated: 100000.0, message: "Use  
Color(cgColor:) when converting a  
CGColor, or create a standard Color  
directly")  
@available(tvOS, introduced: 14.0,  
deprecated: 100000.0, message: "Use  
Color(cgColor:) when converting a  
CGColor, or create a standard Color  
directly")  
@available(watchOS, introduced: 7.0,  
deprecated: 100000.0, message: "Use  
Color(cgColor:) when converting a
```

```
CGColor, or create a standard Color  
directly")  
@available(visionOS, introduced: 1.0,  
deprecated: 100000.0, message: "Use  
Color(cgColor:) when converting a  
CGColor, or create a standard Color  
directly")  
extension Color {  
  
    /// Creates a color from a Core  
    /// Graphics color.  
    ///  
    /// - Parameter color: A  
    ///  
<doc://com.apple.documentation/documentation/CoreGraphics/CGColor> instance  
    /// from which to create a color.  
    public init(_ cgColor: CGColor)  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Color {  
  
    /// Creates a color from a Core  
    /// Graphics color.  
    ///  
    /// - Parameter color: A  
    ///  
<doc://com.apple.documentation/documentation/CoreGraphics/CGColor> instance  
    /// from which to create a color.  
    public init(cgColor: CGColor)
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Color {  
  
    /// Creates a color from a color set  
    /// that you indicate by name.  
    ///  
    /// Use this initializer to load a  
    /// color from a color set stored in an  
    /// Asset Catalog. The system  
    /// determines which color within the set to  
    /// use  
    /// based on the environment at  
    /// render time. For example, you  
    /// can provide light and dark  
    /// versions for background and foreground  
    /// colors:  
    ///  
    /// ! [A screenshot of color sets for  
    /// foreground and background colors,  
    /// each with light and dark  
    /// variants,  
    /// in an Asset Catalog.] (Color-  
    init-1)  
    ///  
    /// You can then instantiate colors  
    /// by referencing the names of the assets:  
    ///  
    ///     struct Hello: View {  
    ///         var body: some View {  
    ///             ZStack {
```

```
    /**
     Color("background")
     /**
     world!")
     /**
     style(Color("foreground")))
     /**
     /**
     .frame(width: 200,
height: 100)
     /**
     }
     /**
     }
     /**
     /**
     /// SwiftUI renders the appropriate
colors for each appearance:
     /**
     /**
     ! [A side by side comparison of
light and dark appearance screenshots
     /**
     of the same content. The light
variant shows dark text on a light
     /**
     background, while the dark
variant shows light text on a dark
     /**
     background.] (Color-init-2)
     /**
     /**
     - Parameters:
     /**
     - name: The name of the color
resource to look up.
     /**
     - bundle: The bundle in which
to search for the color resource.
     /**
     If you don't indicate a
bundle, the initializer looks in your
app's
     /**
     main bundle by default.
public init(_ name: String, bundle:
```

```
Bundle? = nil)
}

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

    /// Initialize a `Color` with a color
    resource.
    public init(_ resource:
    ColorResource)
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Color {

    /// Multiplies the opacity of the
    color by the given amount.
    ///
    /// - Parameter opacity: The amount
    by which to multiply the opacity of the
    /// color.
    /// - Returns: A view with modified
    opacity.
    public func opacity(_ opacity:
    Double) -> Color

    /// Returns a version of self mixed
    with `rhs` by the amount specified
    /// by `fraction`.
    ///
    /// - Parameters:
```

```
    /// - rhs: The color to mix `self`  
    with.  
    /// - fraction: The amount of  
blending, `0.5` means `self` is mixed in  
    /// equal parts with  
`rhs`.  
    /// - colorSpace: The color space  
used to mix the colors.  
    /// - Returns: A new `Color` based on  
`self` and `rhs`.  
    @available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
    public func mix(with rhs: Color, by  
fraction: Double, in colorSpace:  
Gradient.ColorSpace = .perceptual) ->  
Color  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Color.Resolved : ShapeStyle {  
  
    /// The type of shape style this will  
resolve to.  
    ///  
    /// When you create a custom shape  
style, Swift infers this type  
    /// from your implementation of the  
required `resolve` function.  
    @available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
    public typealias Resolved = Never  
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Color.Resolved :  
CustomStringConvertible {  
  
    /// A textual representation of this  
instance.  
    ///  
    /// Calling this property directly is  
discouraged. Instead, convert an  
    /// instance of any type to a string  
by using the `String(describing:)`  
    /// initializer. This initializer  
works with any type, and uses the custom  
    /// `description` property for types  
that conform to  
    /// `CustomStringConvertible`:  
    ///  
    ///     struct Point:  
CustomStringConvertible {  
    ///         let x: Int, y: Int  
    ///  
    ///         var description: String {  
    ///             return "(\(x), \(y))"  
    ///         }  
    ///     }  
    ///  
    ///         let p = Point(x: 21, y: 30)  
    ///         let s = String(describing: p)  
    ///         print(s)  
    ///         // Prints "(21, 30)"  
    ///
```

```
    /// The conversion of `p` to a string
    // in the assignment to `s` uses the
    /// `Point` type's `description`
    property.
    public var description: String {
get }
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Color.Resolved : Animatable {

    /// The type defining the data to
    animate.
    public typealias AnimatableData =
    AnimatablePair<Float,
    AnimatablePair<Float,
    AnimatablePair<Float, Float>>>

    /// The data to animate.
    public var animatableData:
    Color.Resolved.AnimatableData
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Color.Resolved {

    /// Creates a resolved color from
    red, green, and blue component values.
    ///
    /// A standard sRGB color space
    clamps each color component – `red`,
```

```
    /// `green`, and `blue` – to a range
of `0` to `1`, but SwiftUI colors
    /// use an extended sRGB color space,
so
        /// you can use component values
outside that range. This makes it
        /// possible to create colors using
the ``RGBColorSpace/sRGB`` or
        /// ``RGBColorSpace/sRGBLinear``
color space that make full use of the
        /// wider gamut of a display that
supports ``RGBColorSpace/displayP3``.

        ///
        /// - Parameters:
        ///   - colorSpace: The profile that
specifies how to interpret the
        ///   color for display. The
default is ``RGBColorSpace/sRGB``.
        ///   - red: The amount of red in the
color.
        ///   - green: The amount of green in
the color.
        ///   - blue: The amount of blue in
the color.
        ///   - opacity: An optional degree
of opacity, given in the range `0`
        ///   to `1`. A value of `0` means
100% transparency, while a value of
        ///   `1` means 100% opacity. The
default is `1`.

    public init(colorSpace:
Color.RGBColorSpace = .sRGB, red: Float,
green: Float, blue: Float, opacity: Float
```

```
= 1)

    /// The amount of red in the color in
    the sRGB color space.
    public var red: Float

    /// The amount of green in the color
    in the sRGB color space.
    public var green: Float

    /// The amount of blue in the color
    in the sRGB color space.
    public var blue: Float
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Color.Resolved : Codable {

    /// Encodes this value into the given
    encoder.
    ///
    /// If the value fails to encode
    anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
    any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
    to write data to.
    public func encode(to encoder: any
```

```
Encoder) throws
```

```
    /// Creates a new instance by  
    decoding from the given decoder.  
    ///  
    /// This initializer throws an error  
    if reading from the decoder fails, or  
    /// if the data read is corrupted or  
    otherwise invalid.  
    ///  
    /// - Parameter decoder: The decoder  
    to read data from.  
    public init(from decoder: any  
Decoder) throws  
{
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Color.Resolved {  
  
    /// A Core Graphics representation of  
    the color.  
    ///  
    /// You can get a  
    ///  
<doc://com.apple.documentation/documentation/CoreGraphics/CGColor>  
    /// instance from a resolved color.  
    public var cgColor: CGColor { get }  
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)
```

```
extension Color.Resolved :  
BitwiseCopyable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Color.RGBColorSpace : Equatable  
{  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Color.RGBColorSpace : Hashable  
{  
}  
  
/// A matrix to use in an RGBA color  
transformation.  
///  
/// The matrix has five columns, each  
with a red, green, blue, and alpha  
/// component. You can use the matrix for  
tasks like creating a color  
/// transformation  
``GraphicsContext/Filter`` for a  
``GraphicsContext`` using  
/// the  
``GraphicsContext/Filter/colorMatrix(_:)``  
` method.  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
@frozen public struct ColorMatrix :  
Equatable {
```

```
public var r1: Float  
public var r2: Float  
public var r3: Float  
public var r4: Float  
public var r5: Float  
public var g1: Float  
public var g2: Float  
public var g3: Float  
public var g4: Float  
public var g5: Float  
public var b1: Float  
public var b2: Float  
public var b3: Float  
public var b4: Float  
public var b5: Float  
public var a1: Float
```

```
public var a2: Float  
  
public var a3: Float  
  
public var a4: Float  
  
public var a5: Float  
  
/// Creates the identity matrix.  
@inlinable public init()  
  
    /// 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:  
ColorMatrix, b: ColorMatrix) -> Bool  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension ColorMatrix : Sendable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS
```

```
15.0, watchOS 8.0, *)
extension ColorMatrix : BitwiseCopyable {
}

/// The set of possible working color
spaces for color-compositing operations.
///
/// Each color space guarantees the
preservation of a particular range of
color
/// values.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public enum ColorRenderingMode : Sendable
{

    /// The non-linear sRGB working color
space.
    ///
    /// Color component values outside
the range `[0, 1]` produce undefined
    /// results. This color space is
gamma corrected.
    case nonLinear

    /// The linear sRGB working color
space.
    ///
    /// Color component values outside
the range `[0, 1]` produce undefined
    /// results. This color space isn't
gamma corrected.
    case linear
```

```
    /// The extended linear sRGB working
color space.
    ///
    /// Color component values outside
the range `[0, 1]` are preserved.
    /// This color space isn't gamma
corrected.
case extendedLinear

    /// 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:
ColorRenderingMode, b:
ColorRenderingMode) -> Bool

    /// 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)  

  

    /// 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 }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ColorRenderingMode : Equatable
{

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ColorRenderingMode : Hashable {
}

/// The possible color schemes,
/// corresponding to the light and dark
/// appearances.
///
/// You receive a color scheme value when
/// you read the
/// ``EnvironmentValues/colorScheme``
/// environment value. The value tells you if
/// a light or dark appearance currently
/// applies to the view. SwiftUI updates
/// the value whenever the appearance
/// changes, and redraws views that
/// depend on the value. For example, the
/// following ``Text`` view automatically
/// updates when the user enables Dark
/// Mode:
///
```

```
///      @Environment(\.colorScheme)
private var colorScheme
///
///      var body: some View {
///          Text(colorScheme == .dark ?
"Dark" : "Light")
///
/// Set a preferred appearance for a
/// particular view hierarchy to override
/// the user's Dark Mode setting using
/// the ``View/preferredColorScheme(_:)``
/// view modifier.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public enum ColorScheme : CaseIterable,
Sendable {

    /// The color scheme that corresponds
    /// to a light appearance.
    case light

    /// The color scheme that corresponds
    /// to a dark appearance.
    case dark

    /// 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:  
ColorScheme, b: ColorScheme) -> Bool  
  
    /// 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)

```
    /// A type that can represent a
    /// collection of all values of this type.
    @available(iOS 13.0, tvOS 13.0,
    watchOS 6.0, macOS 10.15, *)
    public typealias AllCases =
    [ColorScheme]

    /// A collection of all values of
    /// this type.
    nonisolated public static var
    allCases: [ColorScheme] { get }

    /// 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 }
}

@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
extension ColorScheme : Equatable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ColorScheme : Hashable {
}

/// The contrast between the app's
foreground and background colors.
///
/// You receive a contrast value when you
read the
///
``EnvironmentValues/colorSchemeContrast``
environment value. The value
/// tells you if a standard or increased
contrast currently applies to the view.
/// SwiftUI updates the value whenever
the contrast changes, and redraws
/// views that depend on the value. For
example, the following ``Text`` view
/// automatically updates when the user
enables increased contrast:
///
///
/// @Environment(\.colorSchemeContrast)
private var colorSchemeContrast
///
///     var body: some View {
///         Text(colorSchemeContrast
== .standard ? "Standard" : "Increased")
```

```
///      }
///
/// The user sets the contrast by
/// selecting the Increase Contrast option in
/// Accessibility > Display in System
/// Preferences on macOS, or
/// Accessibility > Display & Text Size
/// in the Settings app on iOS.
/// Your app can't override the user's
/// choice. For
/// information about using color and
/// contrast in your app, see
///
<doc://com.apple.documentation/design/human-interface-guidelines/accessibility#Color-and-effects>
/// in the Human Interface Guidelines.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public enum ColorSchemeContrast : CaseIterable, Sendable {

    /// SwiftUI displays views with
    /// standard contrast between the app's
    /// foreground and background colors.
    case standard

    /// SwiftUI displays views with
    /// increased contrast between the app's
    /// foreground and background colors.
    case increased

    /// 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:
ColorSchemeContrast, b:
ColorSchemeContrast) -> Bool
```

```
/// 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)

    /// A type that can represent a
collection of all values of this type.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias AllCases =
[ColorSchemeContrast]

    /// A collection of all values of
this type.
    nonisolated public static var
allCases: [ColorSchemeContrast] { get }

    /// 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 }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ColorSchemeContrast : Equatable
{

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ColorSchemeContrast : Hashable
{



/// A shape that is replaced by an inset
version of the current
/// container shape. If no container
shape was defined, is replaced by
/// a rectangle.
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
@frozen public struct
ContainerRelativeShape : Shape {

    /// Describes this shape as a path
within a rectangular frame of reference.
    ///
    /// - Parameter rect: The frame of
reference for describing this shape.
```

```
///  
/// - Returns: A path that describes  
this shape.  
nonisolated public func path(in rect:  
CGRect) -> Path  
  
@inlinable public init()  
  
    /// The type defining the data to  
animate.  
    @available(iOS 14.0, tvOS 14.0,  
watchOS 7.0, macOS 11.0, *)  
    public typealias AnimatableData =  
EmptyAnimatableData  
  
    /// The type of view representing the  
body of this view.  
    ///  
    /// When you create a custom view,  
Swift infers this type from your  
    /// implementation of the required  
``View/body-swift.property`` property.  
    @available(iOS 14.0, tvOS 14.0,  
watchOS 7.0, macOS 11.0, *)  
    public typealias Body  
}  
  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
extension ContainerRelativeShape :  
InsettableShape {  
  
    /// Returns `self` inset by `amount`.
```

```
    @inlinable nonisolated public func
    inset(by amount: CGFloat) -> some
    InsettableShape

    /// The type of the inset shape.
    @available(iOS 14.0, tvOS 14.0,
    watchOS 7.0, macOS 11.0, *)
    public typealias InsetShape = some
    InsettableShape
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension ContainerRelativeShape :  

BitwiseCopyable {

/// A key for accessing container values.
///
/// You can create custom container
values by extending the
/// ``ContainerValues`` structure with
new properties.
/// First declare a new container value
key type and specify a value for the
/// required ``defaultValue`` property:
///
///     private struct
MyContainerValueKey: ContainerValueKey {
///         static let defaultValue:
String = "Default value"
///     }
```

```
///  
/// The Swift compiler automatically  
infers the associated ``Value`` type as  
the  
/// type you specify for the default  
value. Then use the key to define a new  
/// container value property:  
///  
///     extension ContainerValues {  
///         var myCustomValue: String {  
///             get  
///             { self[MyContainerValueKey.self] }  
///             set  
///             { self[MyContainerValueKey.self] =  
///               newValue }  
///         }  
///     }  
///  
/// Clients of your container value never  
use the key directly.  
/// Instead, they use the key path of  
your custom container value property.  
/// To set the container value for a  
view, add the  
/// ``View/containerValue(_:_:)`` view  
modifier to that view:  
///  
///     MyView()  
///         .containerValue(\.myCustomVal  
ue, "Another string")  
///  
/// As a convenience, you can also define  
a dedicated view modifier to
```

```
/// apply this container value:  
///  
///     extension View {  
///         func myCustomValue(_  
myCustomValue: String) -> some View {  
///  
containerValue(\.myCustomValue,  
myCustomValue)  
///     }  
///  
/// This improves clarity at the call  
site:  
///  
///     MyView()  
///             .myCustomValue("Another  
string")  
///  
/// To read the container value, use  
`Group(subviews:)` on a containing  
/// view, and then access the container  
value on members of that  
/// collection.  
///  
///     @ViewBuilder var content: some  
View {  
///  
Text("A").myCustomValue("Hello")  
///  
Text("B").myCustomValue("World")  
///     }  
///  
///     Group(subviews: content)  
{ subviews in
```

```
///          ForEach(subviews) { subview
in
///
Text(subview.containerValues.myCustomValue)
///          }
///
/// In practice, this will mostly be used
by views that contain multiple other
/// views to extract information from
their subviews. You could turn the
example
/// above into such a container view as
follows:
///
/// struct MyContainer<Content:
View>: View {
///     var content: Content
///
///     init(@ViewBuilder content: () -> Content) {
///         self.content = content()
///     }
///
///     var body: some View {
///         Group(subviews: content)
{ subviews in
///             ForEach(subviews)
{ subview in
///                 // Display each
view side-by-side with its custom value.
///                 HStack {
```

```
/// subview
///
Text(subview.containerValues.myCustomValue)
/// }
///
/// }
///
/// }
///
/// }
///
/// }
///
/// @available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public protocol ContainerValueKey {
    /// The type of value produced by the
    /// container value.
    associatedtype Value

    /// The default value of the
    /// container value.
    static var defaultValue: Self.Value {
        get
    }

    /// A collection of container values
    /// associated with a given view.
    ///
    /// SwiftUI exposes a collection of
    /// values associated with each view in the
    /// `ContainerValues` structure.
    ///
    /// Container values you set on a given
    /// view affect only that view.
}
```

```
/// Like preferences, container values  
/// can be read by views above the view  
/// they're set on. Unlike preferences,  
however, container values don't have  
/// merging behavior because they don't  
escape their closest container.  
/// The following example shows the  
container value set on the contained  
view,  
/// and then dropped when it reaches the  
containing ``VStack``.  
///  
///     VStack {  
///  
Text("A").containerValue(\.myCustomValue,  
1) // myCustomValue = 1  
///  
Text("B").containerValue(\.myCustomValue,  
2) // myCustomValue = 2  
///         // Container values are  
unaffected by views that aren't  
containers:  
///             Text("C")  
///                     .containerValue(\.myCusto  
mValue, 3)  
///                     .padding() //  
myCustomValue = 3  
///     } // myCustomValue = it's default  
value, values do not escape the container  
///  
/// Even if a stack has only one child,  
container values still wouldn't  
/// be lifted to the `VStack`. Container
```

```
values don't escape a container
/// even if the container has only one
child.
///
/// Create a custom container value by
declaring a new property
/// in an extension to the container
values structure and applying
/// the ``Entry()`` macro to the variable
declaration:
///
///     extension ContainerValues {
///         @Entry var myCustomValue:
String = "Default value"
///     }
///
/// Clients of your value then access the
value by reading it from the container
/// values collection of a ``Subview``.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct ContainerValues {

    /// Accesses the particular container
value associated with a custom key.
    ///
    /// Create a custom container value
by declaring a new property
    /// in an extension to the container
values structure and applying
    /// the ``Entry()`` macro to the
variable declaration:
    ///
```

```
    /// extension ContainerValues {
    ///     @Entry var myCustomValue:
String = "Default value"
    /// }
    ///
    /// You use custom container values
the same way you use system-provided
    /// values, setting a value with the
``View/containerValue(_:_:)`` view
    /// modifier, and reading values from
a ``Subview`` provided by the
    /// `subviews` modifier. You can also
provide a dedicated view modifier as a
    /// convenience for setting the
value:
```

```
    ///
    /// extension View {
    ///     func myCustomValue(_
myCustomValue: String) -> some View {
    ///
containerValue(\.myCustomValue,
myCustomValue)
    ///         }
    ///     }
    ///
public subscript<Key>(key: Key.Type)
-> Key.Value where Key :
ContainerValueKey
```

```
    /// The tag value for the given type
if the container values contains one.
```

```
    ///
    /// Tag values are set using the
```

```
``View/tag`` modifier.  
    ///  
    /// - Parameter type: The type to get  
    the tag value for.  
    /// - Returns: The tag value for the  
    given type if the subview has one,  
    /// otherwise `nil`.  
    public func tag<V>(for type: V.Type)  
-> V? where V : Hashable  
  
    /// Returns true if the container  
    values contain a tag matching a given  
    /// value.  
    ///  
    /// Tag values are set using the  
``View/tag`` modifier.  
    ///  
    /// - Parameter tag: The tag value to  
    check for.  
    /// - Returns: If the container  
    values has a tag matching the given  
    value.  
    public func hasTag<V>(_ tag: V) ->  
Bool where V : Hashable  
}  
  
/// Constants that define how a view's  
content fills the available space.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public enum ContentMode :  
Hashable, CaseIterable {
```

```
    /// An option that resizes the
    content so it's all within the available
    space,
    /// both vertically and horizontally.
    ///
    /// This mode preserves the content's
    aspect ratio.
    /// If the content doesn't have the
    same aspect ratio as the available
    /// space, the content becomes the
    same size as the available space on
    /// one axis and leaves empty space
    on the other.

case fit

    /// An option that resizes the
    content so it occupies all available
    space,
    /// both vertically and horizontally.
    ///
    /// This mode preserves the content's
    aspect ratio.
    /// If the content doesn't have the
    same aspect ratio as the available
    /// space, the content becomes the
    same size as the available space on
    /// one axis, and larger on the other
    axis.

case fill

    /// 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:  
ContentMode, b: ContentMode) -> Bool
```

```
    /// 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)
```

```
    /// A type that can represent a  
collection of all values of this type.
```

```
    @available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
    public typealias AllCases =  
[ContentMode]
```

```
    /// A collection of all values of  
this type.
```

```
    nonisolated public static var  
allCases: [ContentMode] { get }
```

```
    /// 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 hashCode: Int { get }

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ContentMode : Sendable {

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ContentMode : BitwiseCopyable {

}

/// A kind for the content shape of a
view.
///
/// The kind is used by the system to
influence various effects, hit-testing,
/// and more.
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
public struct ContentShapeKinds :
OptionSet, Sendable {

    /// The corresponding value of the
raw type.
    ///
    /// A new instance initialized with
`rawValue` will be equivalent to this
    /// instance. For example:
    ///
    ///     enum PaperSize: String {
    ///         case A4, A5, Letter,
```

## Legal

```
    /**
     */
    /**
     *      let selectedSize =
PaperSize.Letter
    /**
     *      print(selectedSize.rawValue)
    /**
     *      // Prints "Letter"
    /**
     *      print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    /**
     *      // Prints "true"
public var rawValue: Int

    /**
     * Creates a content shape kind.
public init(rawValue: Int)

    /**
     * The kind for hit-testing and
accessibility.
    /**
    /**
     * Setting a content shape with this
kind causes the view to hit-test
    /**
     * using the specified shape.
public static let interaction:
ContentShapeKinds

    /**
     * The kind for drag and drop
previews.
    /**
    /**
     * When using this kind, only the
preview shape is affected. To control the
    /**
     * shape used to hit-test and start
the drag preview, use the `interaction`
```

```
    /// kind.  
    @available(watchOS, unavailable)  
    @available(tvOS, unavailable)  
    public static let dragPreview:  
ContentShapeKinds  
  
    /// The kind for the focus effect.  
    @available(iOS, unavailable)  
    @available(tvOS, unavailable)  
    @available(visionOS, unavailable)  
    public static let focusEffect:  
ContentShapeKinds  
  
    /// The kind for accessibility  
visuals and sorting.  
    ///  
    /// Setting a content shape with this  
kind causes the accessibility frame  
    /// and path of the view's underlying  
accessibility element to match the  
    /// shape without adjusting the hit-  
testing shape, updating the visual focus  
    /// ring that assistive apps, such as  
VoiceOver, draw, as well as how the  
    /// element is sorted. Updating the  
accessibility shape is only required if  
    /// the shape or size used to hit-  
test significantly diverges from the  
visual  
    /// shape of the view.  
    ///  
    /// To control the shape for  
accessibility and hit-testing, use the
```

```
`interaction` kind.  
    @available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
    public static let accessibility:  
ContentShapeKinds  
  
        /// The type of the elements of an  
array literal.  
        @available(iOS 15.0, tvOS 15.0,  
watchOS 8.0, macOS 12.0, *)  
        public typealias ArrayLiteralElement  
= ContentShapeKinds  
  
        /// The element type of the option  
set.  
        ///  
        /// To inherit all the default  
implementations from the `OptionSet`  
protocol,  
        /// the `Element` type must be  
`Self`, the default.  
        @available(iOS 15.0, tvOS 15.0,  
watchOS 8.0, macOS 12.0, *)  
        public typealias Element =  
ContentShapeKinds  
  
        /// The raw type that can be used to  
represent all values of the conforming  
        /// type.  
        ///  
        /// Every distinct value of the  
conforming type has a corresponding  
unique
```

```
    /// value of the `RawValue` type, but
    /// there may be values of the `RawValue`
    /// type that don't have a
    /// corresponding value of the conforming
    /// type.
    @available(iOS 15.0, tvOS 15.0,
    watchOS 8.0, macOS 12.0, *)
    public typealias RawValue = Int
}

@available(iOS, introduced: 13.0,
deprecated: 100000.0, renamed:
"DynamicTypeSize")
@available(macOS, introduced: 10.15,
deprecated: 100000.0, renamed:
"DynamicTypeSize")
@available(tvOS, introduced: 13.0,
deprecated: 100000.0, renamed:
"DynamicTypeSize")
@available(watchOS, introduced: 6.0,
deprecated: 100000.0, renamed:
"DynamicTypeSize")
@available(visionOS, introduced: 1.0,
deprecated: 100000.0, renamed:
"DynamicTypeSize")
public enum ContentSizeCategory :  
    Hashable, CaseIterable {  
  
    case extraSmall  
  
    case small  
  
    case medium
```

```
case large

case extraLarge

case extraExtraLarge

case extraExtraExtraLarge

case accessibilityMedium

case accessibilityLarge

case accessibilityExtraLarge

case accessibilityExtraExtraLarge

case accessibilityExtraExtraExtraLarge

    /// A Boolean value indicating
    whether the content size category is one
    that
        /// is associated with accessibility.
        @available(iOS 13.4, macOS 10.15.4,
        tvOS 13.4, watchOS 6.2, *)
        public var isAccessibilityCategory:
Bool { get }

    /// 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:  
ContentSizeCategory, b:  
ContentSizeCategory) -> Bool

  /// 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)

    /// A type that can represent a
collection of all values of this type.
    @available(iOS, introduced: 13.0,
deprecated: 100000.0, renamed:
"DynamicTypeSize")
    @available(tvOS, introduced: 13.0,
deprecated: 100000.0, renamed:
"DynamicTypeSize")
    @available(watchOS, introduced: 6.0,
deprecated: 100000.0, renamed:
"DynamicTypeSize")
    @available(visionOS, introduced: 1.0,
deprecated: 100000.0, renamed:
"DynamicTypeSize")
    @available(macOS, introduced: 10.15,
deprecated: 100000.0, renamed:
"DynamicTypeSize")
    public typealias AllCases =
[ContentSizeCategory]

    /// A collection of all values of
this type.
    nonisolated public static var
allCases: [ContentSizeCategory] { get }

    /// 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 }
```

```
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
extension ContentSizeCategory {
```

```
    /// Returns a Boolean value  
indicating whether the value of the first  
argument is less than that of the second  
argument.
```

```
    public static func < (lhs:  
ContentSizeCategory, rhs:  
ContentSizeCategory) -> Bool
```

```
    /// Returns a Boolean value  
indicating whether the value of the first  
argument is less than or equal to that of  
the second argument.
```

```
    public static func <= (lhs:
```

```
ContentSizeCategory, rhs:  
ContentSizeCategory) -> Bool
```

```
    /// Returns a Boolean value  
    indicating whether the value of the first  
    argument is greater than that of the  
    second argument.
```

```
    public static func > (lhs:  
ContentSizeCategory, rhs:  
ContentSizeCategory) -> Bool
```

```
    /// Returns a Boolean value  
    indicating whether the value of the first  
    argument is greater than or equal to that  
    of the second argument.
```

```
    public static func >= (lhs:  
ContentSizeCategory, rhs:  
ContentSizeCategory) -> Bool  
}
```

```
/// A kind of transition that applies to  
the content within a single view,  
/// rather than to the insertion or  
removal of a view.
```

```
///
```

```
/// Set the behavior of content  
transitions within a view with the  
/// ``View/contentTransition(_:)``  
modifier, passing in one of the defined  
/// transitions, such as ``opacity`` or  
``interpolate`` as the parameter.
```

```
///
```

```
/// > Tip: Content transitions only take
```

```
effect within transactions that apply
/// an ``Animation`` to the views inside
the ``View/contentTransition(_:)``
/// modifier.
///
/// Content transitions only take effect
within the context of an
/// ``Animation`` block.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public struct ContentTransition :  
Equatable, Sendable {  
  
    /// The identity content transition,
which indicates that content changes
    /// shouldn't animate.
    ///
    /// You can pass this value to a
``View/contentTransition(_:)``
    /// modifier to selectively disable
animations that would otherwise
    /// be applied by a
``withAnimation(_:_:)`` block.
    public static let identity:  
ContentTransition  
  
    /// A content transition that
indicates content fades from transparent
    /// to opaque on insertion, and from
opaque to transparent on removal.
    public static let opacity:  
ContentTransition
```

```
    /// A content transition that
    indicates the views attempt to
    interpolate
        /// their contents during
    transitions, where appropriate.
        ///
        /// Text views can interpolate
    transitions when the text views have
        /// identical strings. Matching glyph
    pairs can animate changes to their
        /// color, position, size, and any
    variable properties. Interpolation can
        /// apply within a ``Font/Design``
    case, but not between cases, or between
        /// entirely different fonts. For
    example, you can interpolate a change
        /// between ``Font/Weight/thin`` and
    ``Font/Weight/black`` variations of a
        /// font, since these are both cases
    of ``Font/Weight``. However, you can't
        /// interpolate between the default
    design of a font and its Italic version,
        /// because these are different
    fonts. Any changes that can't show an
        /// interpolated animation use an
    opacity animation instead.
        ///
        /// Symbol images created with the
    ``Image/init(systemName:)`` initializer
        /// work the same way as text:
    changes within the same symbol attempt to
        /// interpolate the symbol's paths.
    When interpolation is unavailable, the
```

```
    /// system uses an opacity transition  
instead.
```

```
public static let interpolate:  
ContentTransition
```

```
    /// Creates a content transition  
intended to be used with `Text`
```

```
    /// views displaying numeric text. In  
certain environments changes
```

```
    /// to the text will enable a  
nonstandard transition tailored to
```

```
    /// numeric characters that count up  
or down.
```

```
///
```

```
/// - Parameters:
```

```
    /// - countsDown: true if the  
numbers represented by the text
```

```
    ///      are counting downwards.
```

```
///
```

```
    /// - Returns: a new content  
transition.
```

```
public static func  
numericText(countsDown: Bool = false) ->  
ContentTransition
```

```
    /// Creates a content transition  
intended to be used with `Text`
```

```
    /// views displaying numbers.
```

```
///
```

```
    /// The example below creates a text  
view displaying a particular
```

```
    /// value, assigning the same value  
to the associated transition:
```

```
///  
///     Text("\(value)")  
///         .contentTransition(.numer  
icText(value: value))  
///  
/// - Parameters:  
///   - value: the value represented  
by the `Text` view being  
///           animated. The difference  
between the old and new values  
///           when the text changes will be  
used to determine the  
///           animation direction.  
///  
/// - Returns: a new content  
transition.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
public static func numericText(value:  
Double) -> ContentTransition  
  
    /// 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:  
ContentTransition, b: ContentTransition)  
-> Bool  
}  
  
/// The active appearance expected of  
controls in a window.  
///  
/// `ControlActiveState` and  
`EnvironmentValues.controlActiveState`  
are  
/// deprecated, use  
`EnvironmentValues.appearsActive`  
instead.  
@available(iOS, unavailable)  
@available(macCatalyst, introduced: 13.0,  
deprecated: 100000.0, message: "Use  
`EnvironmentValues.appearsActive`  
instead.")  
@available(macOS, introduced: 10.15,  
deprecated: 100000.0, message: "Use  
`EnvironmentValues.appearsActive`  
instead.")  
@available(tvOS, unavailable)  
@available(watchOS, unavailable)  
@available(visionOS, unavailable)  
public enum ControlActiveState :  
Equatable, CaseIterable, Sendable {  
  
    case key  
  
    case active
```

## case inactive

```
    /// 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:  
ControlActiveState, b:  
ControlActiveState) -> Bool
```

```
    /// 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)  
  
    /// A type that can represent a  
collection of all values of this type.  
    @available(iOS, unavailable, message:  
"Use `EnvironmentValues.appearsActive`  
instead.")  
    @available(tvOS, unavailable,  
message: "Use  
`EnvironmentValues.appearsActive`  
instead.")  
    @available(watchOS, unavailable,  
message: "Use  
`EnvironmentValues.appearsActive`  
instead.")  
    @available(visionOS, unavailable,  
message: "Use  
`EnvironmentValues.appearsActive`  
instead.")  
    @available(macOS, introduced: 10.15,  
deprecated: 100000.0, message: "Use  
`EnvironmentValues.appearsActive`  
instead.")
```

```
    @available(macCatalyst, introduced:  
13.0, deprecated: 100000.0, message: "Use  
`EnvironmentValues.appearsActive`  
instead.")
```

```
    public typealias AllCases =  
[ControlActiveState]
```

```
    /// A collection of all values of  
this type.
```

```
    nonisolated public static var  
allCases: [ControlActiveState] { get }
```

```
    /// 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 }
```

```
}
```

```
@available(iOS, unavailable)
```

```
@available(macCatalyst, introduced: 13.0,  
deprecated: 100000.0, message: "Use
```

```
`EnvironmentValues.appearsActive`  
instead.")  
@available(macOS, introduced: 10.15,  
deprecated: 100000.0, message: "Use  
`EnvironmentValues.appearsActive`  
instead.")  
@available(tvOS, unavailable)  
@available(watchOS, unavailable)  
@available(visionOS, unavailable)  
extension ControlActiveState : Hashable {  
}  
  
/// The size classes, like regular or  
small, that you can apply to controls  
/// within a view.  
@available(iOS 15.0, macOS 10.15, watchOS  
9.0, *)  
@available(tvOS, unavailable)  
public enum ControlSize : CaseIterable,  
Sendable {  
  
    /// A control version that is  
minimally sized.  
    case mini  
  
    /// A control version that is  
proportionally smaller size for space-  
constrained views.  
    case small  
  
    /// A control version that is the  
default size.  
    case regular
```

```
    /// A control version that is
    prominently sized.
    @available(macOS 11.0, *)
    case large

    /// A control version that is
    substantially sized. The largest control
    size.
    /// Resolves to ``ControlSize/large`` on platforms other than visionOS.
    @available(iOS 17.0, macOS 14.0,
    watchOS 10.0, visionOS 1.0, *)
    case extraLarge

    /// A collection of all values of
    this type.
    public static var allCases:
    [ControlSize] { get }

    /// 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:
```

```
ControlSize, b: ControlSize) -> Bool
```

```
    /// 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)
```

```
    /// A type that can represent a  
collection of all values of this type.
```

```
    @available(iOS 15.0, watchOS 9.0,  
macOS 10.15, *)
```

```
@available(tvOS, unavailable)
public typealias AllCases =
[ControlSize]

    /// 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 }
}

@available(iOS 15.0, macOS 10.15, watchOS
9.0, *)
@available(tvOS, unavailable)
extension ControlSize : Equatable {
}

@available(iOS 15.0, macOS 10.15, watchOS
9.0, *)
@available(tvOS, unavailable)
extension ControlSize : Hashable {
}
```

```
/// A resolved coordinate space created
by the coordinate space protocol.
///
/// You don't typically use
`CoordinateSpace` directly. Instead, use
the static
/// properties and functions of
`CoordinateSpaceProtocol` such as
`.global`,
/// `.local`, and `.named(_:)`.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public enum CoordinateSpace {

    /// The global coordinate space at
    // the root of the view hierarchy.
    case global

    /// The local coordinate space of the
    // current view.
    case local

    /// A named reference to a view's
    // local coordinate space.
    case named(AnyHashable)
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension CoordinateSpace {

    public var isGlobal: Bool { get }
```

```
    public var isLocal: Bool { get }
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
```

```
extension CoordinateSpace : Equatable,
```

```
Hashable {
```

```
    /// 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 == (lhs: CoordinateSpace, rhs: CoordinateSpace) -> 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 frame of reference within the  
layout system.
```

```
///
```

```
/// All geometric properties of a view,  
including size, position, and
```

```
/// transform, are defined within the  
local coordinate space of the view's
```

```
/// parent. These values can be converted  
into other coordinate spaces
```

```
/// by passing types conforming to this  
protocol into functions such as
```

```
/// `GeometryProxy.frame(in:)`.
```

```
///
```

```
/// For example, a named coordinate space  
allows you to convert the frame
```

```
/// of a view into the local coordinate  
space of an ancestor view by defining
```

```
/// a named coordinate space using the  
`coordinateSpace(_:)` modifier, then
```

```
/// passing that same named coordinate  
space into the `frame(in:)` function.
```

```
///
```

```
///     VStack {
```

```
///         GeometryReader
```

```
{ geometryProxy in
```

```
///             let distanceFromTop =  
geometryProxy.frame(in:
```

```
"container").origin.y
```

```
///             Text("This view is \
```

```
(distanceFromTop) points from the top of
the VStack")
    }
    .padding()
}
.coordinateSpace(.named("containe
r"))
///
/// You don't typically create types
conforming to this protocol yourself.
/// Instead, use the system-provided
`.global`, `.local`, and `.`named(_:)`'
/// implementations.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public protocol CoordinateSpaceProtocol {

    /// The resolved coordinate space.
    var coordinateSpace: CoordinateSpace
    { get }
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension CoordinateSpaceProtocol where
Self == NamedCoordinateSpace {

    /// Creates a named coordinate space
using the given value.
    ///
    /// Use the `coordinateSpace(_:)`'
modifier to assign a name to the local
    /// coordinate space of a parent
```

```
view. Child views can then refer to that
    /// coordinate space using
`.named(_:)`.
    ///
    /// - Parameter name: A unique value
that identifies the coordinate space.
    ///
    /// - Returns: A named coordinate
space identified by the given value.
public static func named(_ name: some
Hashable) -> NamedCoordinateSpace
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension CoordinateSpaceProtocol where
Self == LocalCoordinateSpace {

    /// The local coordinate space of the
current view.
    public static var local:
LocalCoordinateSpace { get }
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension CoordinateSpaceProtocol where
Self == GlobalCoordinateSpace {

    /// The global coordinate space at
the root of the view hierarchy.
    public static var global:
GlobalCoordinateSpace { get }
}
```

```
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension CoordinateSpaceProtocol where  
Self == NamedCoordinateSpace {  
  
    /// The named coordinate space that  
    is added by the system for the innermost  
    /// containing scroll view that  
    allows scrolling along the provided axis.  
    public static func scrollView(axis:  
Axis) -> Self  
  
    /// The named coordinate space that  
    is added by the system for the innermost  
    /// containing scroll view.  
    public static var scrollView:  
NamedCoordinateSpace { get }  
}  
  
/// A keyframe that uses a cubic curve to  
smoothly interpolate between values.  
///  
/// If you don't specify a start or end  
velocity, SwiftUI automatically  
/// computes a curve that maintains  
smooth motion between keyframes.  
///  
/// Adjacent cubic keyframes result in a  
Catmull–Rom spline.  
///  
/// If a cubic keyframe follows a
```

```
different type of keyframe, such as a
linear
/// keyframe, the end velocity of the
segment defined by the previous keyframe
/// will be used as the starting
velocity.
///
/// Likewise, if a cubic keyframe is
followed by a different type of keyframe,
/// the initial velocity of the next
segment is used as the end velocity of
the
/// segment defined by this keyframe.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct CubicKeyframe<Value> :
KeyframeTrackContent where Value :
Animatable {

    /// Creates a new keyframe using the
given value and timestamp.
    ///
    /// - Parameters:
    ///   - to: The value of the
keyframe.
    ///   - startVelocity: The velocity
of the value at the beginning of the
    ///     segment, or `nil` to
automatically compute the velocity to
maintain
    ///     smooth motion.
    ///   - endVelocity: The velocity of
the value at the end of the segment,
```

```
    /// or `nil` to automatically
compute the velocity to maintain smooth
    /// motion.
    /// - duration: The duration of the
segment defined by this keyframe.
    public init(_ to: Value, duration:
TimeInterval, startVelocity: Value? =
nil, endVelocity: Value? = nil)

    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Body =
CubicKeyframe<Value>
}

/// A type that defines how an animatable
value changes over time.
///
/// Use this protocol to create a type
that changes an animatable value over
/// time, which produces a custom visual
transition of a view. For example, the
/// follow code changes an animatable
value using an elastic ease-in ease-out
/// function:
///
/// struct
ElasticEaseInEaseOutAnimation:
CustomAnimation {
    ///     let duration: TimeInterval
    ///
    ///     func animate<V>(value: V,
time: TimeInterval, context: inout
```

```
AnimationContext<V>) -> V? where V :  
VectorArithmetic {  
    /// if time > duration  
    { return nil } // The animation has  
    finished.  
    ///  
    /// let p = time / duration  
    /// let s = sin((20 * p -  
    11.125) * ((2 * Double.pi) / 4.5))  
    /// if p < 0.5 {  
    ///     return  
    value.scaled(by: -(pow(2, 20 * p - 10) *  
    s) / 2)  
    /// } else {  
    ///     return  
    value.scaled(by: (pow(2, -20 * p + 10) *  
    s) / 2 + 1)  
    /// }  
    /// }  
    /// }  
    /// > Note: To maintain state during the  
    life span of a custom animation, use  
    /// the ``AnimationContext/state``  
    property available on the `context`  
    /// parameter value. You can also use  
    context's  
    /// ``AnimationContext/environment``  
    property to retrieve environment values  
    /// from the view that created the custom  
    animation. For more information, see  
    /// ``AnimationContext``.  
    ///
```

```
/// To create an ``Animation`` instance  
of a custom animation, use the  
/// ``Animation/init(_:)`` initializer,  
passing in an instance of a custom  
/// animation; for example:  
///  
///  
/// Animation(ElasticEaseInEaseOutAnimation(d  
uration: 5.0))  
///  
/// To help make view code more readable,  
extend ``Animation`` and add a static  
/// property and function that returns an  
`Animation` instance of a custom  
/// animation. For example, the following  
code adds the static property  
/// `elasticEaseInEaseOut` that returns  
the elastic ease-in ease-out animation  
/// with a default duration of `0.35`  
seconds. Next, the code adds a method  
/// that returns the animation with a  
specified duration.  
///  
///     extension Animation {  
///         static var  
elasticEaseInEaseOut: Animation  
{ elasticEaseInEaseOut(duration: 0.35) }  
///         static func  
elasticEaseInEaseOut(duration:  
TimeInterval) -> Animation {  
///  
Animation(ElasticEaseInEaseOutAnimation(d  
uration: duration))
```

```
///      }
///  }
///
/// To animate a view with the elastic
/// ease-in ease-out animation, a view calls
/// either `.elasticEaseInEaseOut` or
/// `.`elasticEaseInEaseOut(duration:)`. For
/// example, the follow code includes an
/// Animate button that, when clicked,
/// animates a circle as it moves from
/// one edge of the view to the other,
/// using the elastic ease-in ease-out
/// animation with a duration of `5`
/// seconds:
///
///     struct ElasticEaseInEaseOutView:
View {
    ///         @State private var isActive =
    false
    ///
    ///         var body: some View {
    ///             VStack(alignment:
    isActive ? .trailing : .leading) {
    ///                 Circle()
    ///                     .frame(width:
    100.0)
    ///                     .foregroundColor(
    .accentColor)
    ///
    ///             Button("Animate") {
    ///
    withAnimation(.elasticEaseInEaseOut(durat
    ion: 5.0)) {
```

```
///  
isActive.toggle()  
///  
///  
///  
/// frame(maxWidth: .infinity)  
/// }  
/// padding()  
/// }  
/// }  
///  
/// @Video(source: "animation-20-  
elastic.mp4", poster: "animation-20-  
elastic.png", alt: "A video that shows a  
circle that moves from one edge of the  
view to the other using an elastic ease-  
in ease-out animation. The circle's  
initial position is near the leading edge  
of the view. The circle begins moving  
slightly towards the leading, then  
towards trail edges of the view before it  
moves off the leading edge showing only  
two-thirds of the circle. The circle then  
moves quickly to the trailing edge of the  
view, going slightly beyond the edge so  
that only two-thirds of the circle is  
visible. The circle bounces back into  
full view before settling into position  
near the trailing edge of the view. The  
circle repeats this animation in reverse,  
going from the trailing edge of the view  
to the leading edge.")  
@available(iOS 17.0, macOS 14.0, tvOS
```

```
17.0, watchOS 10.0, *)
@preconcurrency public protocol
CustomAnimation : Hashable, Sendable {

    /// Calculates the value of the
    animation at the specified time.
    ///
    /// Implement this method to
    calculate and return the value of the
    /// animation at a given point in
    time. If the animation has finished,
    /// return `nil` as the value. This
    signals to the system that it can
    /// remove the animation.
    ///
    /// If your custom animation needs to
    maintain state between calls to the
    /// `animate(value:time:context:)`-
    method, store the state data in
    /// `context`. This makes the data
    available to the method next time
    /// the system calls it. To learn
    more about managing state data in a
    /// custom animation, see
    ``AnimationContext``.

    ///
    /// - Parameters:
    ///   - value: The vector to animate
    towards.
    ///   - time: The elapsed time since
    the start of the animation.
    ///   - context: An instance of
    ``AnimationContext`` that provides access
```

```
    /// to state and the animation
    environment.
    /// - Returns: The current value of
    the animation, or `nil` if the
    /// animation has finished.
    nonisolated func animate<V>(value: V,
time: TimeInterval, context: inout
AnimationContext<V>) -> V? where V :
VectorArithmetic

    /// Calculates the velocity of the
    animation at a specified time.
    ///
    /// Implement this method to provide
    the velocity of the animation at a
    /// given time. Should subsequent
    animations merge with the animation,
    /// the system preserves continuity
    of the velocity between animations.
    ///
    /// The default implementation of
    this method returns `nil`.
    ///
    /// > Note: State and environment
    data is available to this method via the
    /// `context` parameter, but
    `context` is read-only. This behavior is
    /// different than with
    ``animate(value:time:context)``
    and
    ///
    ``shouldMerge(previous:value:time:context
:)`` where `context` is
    /// an `inout` parameter, letting you
```

```
change the context including state
    /// data of the animation. For more
    information about managing state data
    /// in a custom animation, see
``AnimationContext``.

    ///
    /// - Parameters:
    ///   - value: The vector to animate
    towards.
    ///   - time: The amount of time
    since the start of the animation.
    ///   - context: An instance of
``AnimationContext`` that provides access
    /// to state and the animation
    environment.
    /// - Returns: The current velocity
    of the animation, or `nil` if the
    /// animation has finished.
nonisolated func velocity<V>(value:
V, time: TimeInterval, context:
AnimationContext<V>) -> V? where V :
VectorArithmetic

    /// Determines whether an instance of
    the animation can merge with other
    /// instance of the same type.
    ///
    /// When a view creates a new
    animation on an animatable value that
    already
    /// has a running animation of the
    same animation type, the system calls
    /// the
```

```
`shouldMerge(previous:value:time:context:  
)` method on the new  
    /// instance to determine whether it  
can merge the two instance. Implement  
    /// this method if the animation can  
merge with another instance. The  
    /// default implementation returns  
`false`.  
    ///  
    /// If  
`shouldMerge(previous:value:time:context:  
)` returns `true`, the  
    /// system merges the new animation  
instance with the previous animation.  
    /// The system provides to the new  
instance the state and elapsed time from  
    /// the previous one. Then it removes  
the previous animation.  
    ///  
    /// If this method returns `false`,  
the system doesn't merge the animation  
    /// with the previous one. Instead,  
both animations run together and the  
    /// system combines their results.  
    ///  
    /// If your custom animation needs to  
maintain state between calls to the  
    ///  
`shouldMerge(previous:value:time:context:  
)` method, store the state  
    /// data in `context`. This makes the  
data available to the method next  
    /// time the system calls it. To
```

learn more, see ``AnimationContext``.

```
    /**
     * - Parameters:
     *   - previous: The previous
     *     running animation.
     *   - value: The vector to animate
     *     towards.
     *   - time: The amount of time
     *     since the start of the previous
     *     animation.
     *   - context: An instance of
     *     ``AnimationContext`` that provides access
     *     to state and the animation
     *     environment.
     * - Returns: A Boolean value of
     *   `true` if the animation should merge with
     *   the previous animation;
     * otherwise, `false`.
    nonisolated func
shouldMerge<V>(previous: Animation,
value: V, time: TimeInterval, context:
inout AnimationContext<V>) -> Bool where
V : VectorArithmetic
}

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

    /**
     * Calculates the velocity of the
     * animation at a specified time.
     */
    /**
     * Implement this method to provide

```

```
the velocity of the animation at a
    /// given time. Should subsequent
animations merge with the animation,
    /// the system preserves continuity
of the velocity between animations.
    ///
    /// The default implementation of
this method returns `nil`.
    ///
    /// > Note: State and environment
data is available to this method via the
    /// `context` parameter, but
`context` is read-only. This behavior is
    /// different than with
``animate(value:time:context:)`` and
    ///
``shouldMerge(previous:value:time:context
:)`` where `context` is
    /// an `inout` parameter, letting you
change the context including state
    /// data of the animation. For more
information about managing state data
    /// in a custom animation, see
``AnimationContext``.
    ///
    /// - Parameters:
    ///   - value: The vector to animate
towards.
    ///   - time: The amount of time
since the start of the animation.
    ///   - context: An instance of
``AnimationContext`` that provides access
    /// to state and the animation
```

environment.

    /// – Returns: The current velocity  
    of the animation, or `nil` if the

    /// animation has finished.

```
public func velocity<V>(value: V,  
time: TimeInterval, context:  
AnimationContext<V>) -> V? where V :  
VectorArithmetic
```

    /// Determines whether an instance of  
the animation can merge with other

    /// instance of the same type.

    ///

    /// When a view creates a new  
animation on an animatable value that  
already

    /// has a running animation of the  
same animation type, the system calls

    /// the

```
`shouldMerge(previous:value:time:context:  
)` method on the new
```

    /// instance to determine whether it  
can merge the two instance. Implement

    /// this method if the animation can  
merge with another instance. The

    /// default implementation returns

    `false`.

    ///

    /// If

```
`shouldMerge(previous:value:time:context:  
)` returns `true`, the
```

    /// system merges the new animation  
instance with the previous animation.

```
    /// The system provides to the new
    instance the state and elapsed time from
    /// the previous one. Then it removes
    the previous animation.
    ///
    /// If this method returns `false`,
    the system doesn't merge the animation
    /// with the previous one. Instead,
    both animations run together and the
    /// system combines their results.
    ///
    /// If your custom animation needs to
    maintain state between calls to the
    ///
    `shouldMerge(previous:value:time:context:
)` method, store the state
    /// data in `context`. This makes the
    data available to the method next
    /// time the system calls it. To
    learn more, see ``AnimationContext``.
    ///
    /// - Parameters:
    ///   - previous: The previous
    running animation.
    ///   - value: The vector to animate
    towards.
    ///   - time: The amount of time
    since the start of the previous
    animation.
    ///   - context: An instance of
    ``AnimationContext`` that provides access
    ///   to state and the animation
    environment.
```

```
    /// - Returns: A Boolean value of  
`true` if the animation should merge with  
    /// the previous animation;  
otherwise, `false`.
```

```
    public func shouldMerge<V>(previous:  
Animation, value: V, time: TimeInterval,  
context: inout AnimationContext<V>) ->  
Bool where V : VectorArithmetic  
}
```

```
/// A selectability type that disables  
text selection by the person using your  
app.
```

```
///
```

```
/// Don't use this type directly.  
Instead, use
```

```
``TextSelectability/disabled``.  
@available(iOS 15.0, macOS 12.0, *)  
@available(tvOS, unavailable)  
@available(watchOS, unavailable)  
public struct DisabledTextSelectability :  
TextSelectability {
```

```
    /// A Boolean value that indicates  
whether the selectability type allows  
    /// selection.
```

```
    ///
```

```
    /// Conforming types, such as  
``EnabledTextSelectability`` and  
    /// ``DisabledTextSelectability``,  
return `true` or `false` for this  
    /// property as appropriate. SwiftUI  
expects this value for a given
```

```
    /// selectability type to be
    constant, unaffected by global state.
    public static let allowsSelection:
Bool
}

/// An interface for a stored variable
that updates an external property of a
/// view.
///
/// The view gives values to these
properties prior to recomputing the
view's
/// ``View/body-swift.property``.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public protocol DynamicProperty {

    /// Updates the underlying value of
the stored value.
    ///
    /// SwiftUI calls this function
before rendering a view's
    /// ``View/body-swift.property`` to
ensure the view has the most recent
    /// value.
    mutating func update()
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension DynamicProperty {
```

```
    /// Updates the underlying value of
    // the stored value.
    ///
    /// SwiftUI calls this function
    before rendering a view's
    /// ``View/body-swift.property`` to
    ensure the view has the most recent
    /// value.
    public mutating func update()
}

/// A Dynamic Type size, which specifies
// how large scalable content should be.
///
/// For more information, see
///
<doc://com.apple.documentation/design/hum
an-interface-guidelines/typography>
/// in the Human Interface Guidelines.
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
public enum DynamicTypeSize : Hashable,
Comparable, CaseIterable, Sendable {

    /// An extra small size.
    case xSmall

    /// A small size.
    case small

    /// A medium size.
    case medium
```

```
    /// A large size.  
    case large  
  
    /// An extra large size.  
    case xLarge  
  
    /// An extra extra large size.  
    case xxLarge  
  
    /// An extra extra extra large size.  
    case xxxLarge  
  
    /// The first accessibility size.  
    case accessibility1  
  
    /// The second accessibility size.  
    case accessibility2  
  
    /// The third accessibility size.  
    case accessibility3  
  
    /// The fourth accessibility size.  
    case accessibility4  
  
    /// The fifth accessibility size.  
    case accessibility5  
  
    /// A Boolean value indicating  
    whether the size is one that is  
    associated  
    /// with accessibility.  
    public var isAccessibilitySize: Bool  
    { get }
```

```
    /// 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:  
DynamicTypeSize, b: DynamicTypeSize) ->  
Bool
```

```
    /// 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 the value of the first  
    /// argument is less than that of the  
second argument.
```

```
    ///
```

```
    /// This function is the only  
requirement of the `Comparable` protocol.  
The
```

```
    /// remainder of the relational  
operator functions are implemented by the  
    /// standard library for any type  
that conforms to `Comparable`.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - lhs: A value to compare.
```

```
    /// - rhs: Another value to  
compare.
```

```
public static func < (a:  
DynamicTypeSize, b: DynamicTypeSize) ->  
Bool
```

```
    /// A type that can represent a
```

```
collection of all values of this type.  
    @available(iOS 15.0, tvOS 15.0,  
watchOS 8.0, macOS 12.0, *)  
    public typealias AllCases =  
[DynamicTypeSize]  
  
    /// A collection of all values of  
this type.  
    nonisolated public static var  
allCases: [DynamicTypeSize] { get }  
  
    /// 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 type of view that generates views  
from an underlying collection of data.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
public protocol DynamicViewContent : View
{
    /// The type of the underlying
    collection of data.
    associatedtype Data : Collection

    /// The collection of underlying
    data.
    var data: Self.Data { get }
}

/// An enumeration to indicate one edge
of a rectangle.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public enum Edge : Int8,
CaseIterable {
    case top
    case leading
    case bottom
    case trailing

    /// An efficient set of edges.
    @frozen public struct Set : OptionSet
    {
        /// The element type of the
        option set.
    }
}
```

```
    /**
     * To inherit all the default
     * implementations from the `OptionSet`
     * protocol,
     *   /// the `Element` type must be
     * `Self`, the default.
     *   public typealias Element =
Edge.Set

        /// The corresponding value of
the raw type.
        /**
         * A new instance initialized
with `rawValue` will be equivalent to
this
        /// instance. For example:
        /**
        /**
         * enum PaperSize: String {
         *   case A4, A5, Letter,
Legal
         */
         }
        /**
        /**
         * let selectedSize =
PaperSize.Letter
        /**
print(selectedSize.rawValue)
        /**
         * // Prints "Letter"
        /**
        /**
         * print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
        /**
         * // Prints "true"
public let rawValue: Int8
```

```
    /// Creates a new option set from
    the given raw value.
    /**
     * This initializer always
     succeeds, even if the value passed as
     `rawValue` exceeds the static properties
     declared as part of the option set. This
     example creates an instance
     of `ShippingOptions` with a raw value
     beyond
     * the highest element, with a
     bit mask that effectively contains all
     the
     * declared static members.
     */
     * let extraOptions =
ShippingOptions(rawValue: 255)
     */
print(extraOptions.isStrictSuperset(of: .
all))
     * // Prints "true"
     */
     * - Parameter rawValue: The raw
value of the option set to create. Each
bit
     * of `rawValue` potentially
represents an element of the option set,
     * though raw values may
include bits that are not defined as
distinct
     * values of the `OptionSet`
```

type.

```
    public init(rawValue: Int8)

    public static let top: Edge.Set
    public static let leading: Edge.Set
    public static let bottom: Edge.Set
    public static let trailing: Edge.Set
    public static let all: Edge.Set
    public static let horizontal: Edge.Set
    public static let vertical: Edge.Set

    /// Creates set of edges
    containing only the specified edge.
    public init(_ e: Edge)

    /// The type of the elements of
    an array literal.
    @available(iOS 13.0, tvOS 13.0,
    watchOS 6.0, macOS 10.15, *)
    public typealias
    ArrayLiteralElement = Edge.Set.Element
```

```
    /// The raw type that can be used
    to represent all values of the conforming
    /// type.
    ///
    /// Every distinct value of the
    conforming type has a corresponding
    unique
    /// value of the `RawValue` type,
    but there may be values of the `RawValue`
    /// type that don't have a
    corresponding value of the conforming
    type.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias RawValue = Int8
}

/// Creates a new instance with the
specified raw value.
///
/// If there is no value of the type
that corresponds with the specified raw
/// value, this initializer returns
`nil`. For example:
///
///     enum PaperSize: String {
///         case A4, A5, Letter,
Legal
///     }
///
///     print(PaperSize(rawValue:
"Legal"))
///     // Prints
```

```
"Optional("PaperSize.Legal")"
    /**
     /**
      print(PaperSize(rawValue:
"Tabloid"))
     /**
      // Prints "nil"
     /**
      /**
      - Parameter rawValue: The raw
      value to use for the new instance.
  public init?(rawValue: Int8)

      /**
      A type that can represent a
      collection of all values of this type.
  @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
  public typealias AllCases = [Edge]

      /**
      The raw type that can be used to
      represent all values of the conforming
      /**
      type.
  /**
      /**
      Every distinct value of the
      conforming type has a corresponding
      unique
      /**
      value of the `RawValue` type, but
      there may be values of the `RawValue`-
      /**
      type that don't have a
      corresponding value of the conforming
      type.
  @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
  public typealias RawValue = Int8

      /**
      A collection of all values of
```

this type.

```
nonisolated public static var
allCases: [Edge] { get }

    /// The corresponding value of the
raw type.
    /**
     * A new instance initialized with
`rawValue` will be equivalent to this
     * instance. For example:
    /**
    /**
        enum PaperSize: String {
    /**
            case A4, A5, Letter,
Legal
    /**
    /**
        let selectedSize =
PaperSize.Letter
    /**
            print(selectedSize.rawValue)
    /**
            // Prints "Letter"
    /**
            print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    /**
            // Prints "true"
    public var rawValue: Int8 { get }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Edge : Equatable {
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Edge : Hashable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Edge : RawRepresentable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Edge : Sendable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Edge : BitwiseCopyable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Edge.Set : BitwiseCopyable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Edge.Set : Sendable {  
}  
  
/// The inset distances for the sides of  
a rectangle.  
@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
@frozen public struct EdgeInsets : Equatable {

    public var top: CGFloat
    public var leading: CGFloat
    public var bottom: CGFloat
    public var trailing: CGFloat

    @inlinable public init(top: CGFloat, leading: CGFloat, bottom: CGFloat, trailing: CGFloat)

    @inlinable public init()

        /// 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: EdgeInsets, b: EdgeInsets) -> Bool
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EdgeInsets : Animatable {  
  
    /// The type defining the data to  
    animate.  
    public typealias AnimatableData =  
        AnimatablePair<CGFloat,  
        AnimatablePair<CGFloat,  
        AnimatablePair<CGFloat, CGFloat>>>  
  
    /// The data to animate.  
    public var animatableData:  
        EdgeInsets.AnimatableData  
    }  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EdgeInsets : Sendable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EdgeInsets : BitwiseCopyable {  
}  
  
/// An ellipse aligned inside the frame  
of the view containing it.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct Ellipse : Shape {
```

```
    /// Describes this shape as a path
    // within a rectangular frame of reference.
    ///
    /// - Parameter rect: The frame of
    // reference for describing this shape.
    ///
    /// - Returns: A path that describes
    // this shape.
    nonisolated public func path(in rect:
CGRect) -> Path

    /// Creates a new ellipse shape.
    @inlinable public init()

    /// Returns the behavior this shape
    // should use for different layout
    /// directions.
    ///
    /// If the layoutDirectionBehavior
    // for a Shape is one that mirrors, the
    /// shape's path will be mirrored
    horizontally when in the specified layout
    /// direction. When mirrored, the
    individual points of the path will be
    /// transformed.
    ///
    /// Defaults to `mirrors` when
    // deploying on iOS 17.0, macOS 14.0,
    /// tvOS 17.0, watchOS 10.0 and
    later, and to `fixed` if not.
    /// To mirror a path when deploying
    to earlier releases, either use
    ///
```

```
`View.flipsForRightToLeftLayoutDirection`  
for a filled or stroked  
    /// shape or conditionally mirror the  
    points in the path of the shape.  
    @available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
    nonisolated public var  
layoutDirectionBehavior:  
LayoutDirectionBehavior { get }  
  
    /// The type defining the data to  
animate.  
    @available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
    public typealias AnimatableData =  
EmptyAnimatableData  
  
    /// The type of view representing the  
body of this view.  
    ///  
    /// When you create a custom view,  
Swift infers this type from your  
    /// implementation of the required  
``View/body-swift.property`` property.  
    @available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
    public typealias Body  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Ellipse : InsettableShape {
```

```
    /// Returns `self` inset by `amount`.  
    @inlinable nonisolated public func  
    inset(by amount: CGFloat) -> some  
    InsettableShape
```

```
    /// The type of the inset shape.  
    @available(iOS 13.0, tvOS 13.0,  
    watchOS 6.0, macOS 10.15, *)  
    public typealias InsetShape = some  
    InsettableShape  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Ellipse : BitwiseCopyable {  
}
```

```
    /// A radial gradient that draws an  
    ellipse.  
    ///  
    /// The gradient maps its coordinate  
    space to the unit space square  
    /// in which its center and radii are  
    defined, then stretches that  
    /// square to fill its bounding rect,  
    possibly also stretching the  
    /// circular gradient to have elliptical  
    contours.  
    ///  
    /// For example, an elliptical gradient  
    centered on the view, filling  
    /// its bounds:
```

```
///  
///  
EllipticalGradient(gradient: .init(colors  
: [.red, .yellow]))  
///  
/// When using an elliptical gradient as  
a shape style, you can also use  
///  
``ShapeStyle/ellipticalGradient(_:center:  
startRadiusFraction:endRadiusFraction:)``  
-  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
@frozen public struct  
EllipticalGradient : ShapeStyle, View,  
Sendable {  
  
    /// Creates an elliptical gradient.  
    ///  
    /// For example, an elliptical  
gradient centered on the top-leading  
    /// corner of the view:  
    ///  
    ///     EllipticalGradient(  
    ///         gradient: .init(colors:  
[.blue, .green]),  
    ///         center: .topLeading,  
    ///         startRadiusFraction: 0,  
    ///         endRadiusFraction: 1)  
    ///  
    /// - Parameters:  
    /// - gradient: The colors and their  
parametric locations.
```

```
    /// - center: The center of the
    circle, in [0, 1] coordinates.
    /// - startRadiusFraction: The start
    radius value, as a fraction
        /// between zero and one. Zero
        maps to the center point, one
        /// maps to the diameter of the
        unit circle.
    /// - endRadiusFraction: The end
    radius value, as a fraction
        /// between zero and one. Zero
        maps to the center point, one
        /// maps to the diameter of the
        unit circle.
    public init(gradient: Gradient,
    center: UnitPoint = .center,
    startRadiusFraction: CGFloat = 0,
    endRadiusFraction: CGFloat = 0.5)
```

```
    /// Creates an elliptical gradient
    from a collection of colors.
    ///
    /// For example, an elliptical
    gradient centered on the top-leading
    /// corner of the view:
    ///
    ///     EllipticalGradient(
    ///         colors: [.blue, .green],
    ///         center: .topLeading,
    ///         startRadiusFraction: 0,
    ///         endRadiusFraction: 1)
    ///
    /// - Parameters:
```

```
    /// - colors: The colors, evenly
    /// distributed throughout the gradient.
    /// - center: The center of the
    /// circle, in [0, 1] coordinates.
    /// - startRadiusFraction: The start
    /// radius value, as a fraction
    /// between zero and one. Zero
    /// maps to the center point, one
    /// maps to the diameter of the
    /// unit circle.
    /// - endRadiusFraction: The end
    /// radius value, as a fraction
    /// between zero and one. Zero
    /// maps to the center point, one
    /// maps to the diameter of the
    /// unit circle.
    public init(colors: [Color], center:
UnitPoint = .center, startRadiusFraction:
CGFloat = 0, endRadiusFraction: CGFloat =
0.5)

    /// Creates an elliptical gradient
    /// from a collection of color stops.
    ///
    /// For example, an elliptical
    /// gradient centered on the top-leading
    /// corner of the view, with some
    /// extra green area:
    ///
    ///     EllipticalGradient(
    ///         stops: [
    ///             .init(color: .blue,
location: 0.0),
```

```
    ///          .init(color: .green,
location: 0.9),
    ///          .init(color: .green,
location: 1.0),
    ///          ],
    ///          center: .topLeading,
    ///          startRadiusFraction: 0,
    ///          endRadiusFraction: 1)

    /// - Parameters:
    /// - stops: The colors and their
parametric locations.
    /// - center: The center of the
circle, in [0, 1] coordinates.
    /// - startRadiusFraction: The start
radius value, as a fraction
    /// between zero and one. Zero
maps to the center point, one
    /// maps to the diameter of the
unit circle.
    /// - endRadiusFraction: The end
radius value, as a fraction
    /// between zero and one. Zero
maps to the center point, one
    /// maps to the diameter of the
unit circle.

public init(stops: [Gradient.Stop],
center: UnitPoint = .center,
startRadiusFraction: CGFloat = 0,
endRadiusFraction: CGFloat = 0.5)

    /// The type of view representing the
body of this view.
```

```
///  
/// When you create a custom view,  
Swift infers this type from your  
/// implementation of the required  
``View/body-swift.property`` property.  
@available(iOS 15.0, tvOS 15.0,  
watchOS 8.0, macOS 12.0, *)  
public typealias Body  
  
/// The type of shape style this will  
resolve to.  
///  
/// When you create a custom shape  
style, Swift infers this type  
/// from your implementation of the  
required `resolve` function.  
@available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
public typealias Resolved = Never  
}  
  
/// An empty type for animatable data.  
///  
/// This type is suitable for use as the  
`animatableData` property of  
/// types that do not have any animatable  
properties.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct EmptyAnimatableData  
: VectorArithmetic {  
  
    @inlinable public init()  
}
```

```
    /// The zero value.  
    ///  
    /// Zero is the identity element for  
    addition. For any value,  
    /// `x + .zero == x` and `.zero + x  
== x`.  
    @inlinable public static var zero:  
EmptyAnimatableData { get }  
  
    /// Adds two values and stores the  
    result in the left-hand-side variable.  
    ///  
    /// - Parameters:  
    ///   - lhs: The first value to add.  
    ///   - rhs: The second value to add.  
    @inlinable public static func +=  
(lhs: inout EmptyAnimatableData, rhs:  
EmptyAnimatableData)  
  
    /// Subtracts the second value from  
    the first and stores the difference in  
    the  
    /// left-hand-side variable.  
    ///  
    /// - Parameters:  
    ///   - lhs: A numeric value.  
    ///   - rhs: The value to subtract  
from `lhs`.  
    @inlinable public static func -=  
(lhs: inout EmptyAnimatableData, rhs:  
EmptyAnimatableData)
```

```
    /// Adds two values and produces
    their sum.
    ///
    /// The addition operator (`+`)
calculates the sum of its two arguments.
For
    /// example:
    ///
    ///      1 + 2          // 3
    ///      -10 + 15        // 5
    ///      -15 + -5         //
-20
    ///      21.5 + 3.25      //
24.75
    ///
    /// You cannot use `+` with arguments
of different types. To add values of
    /// different types, convert one of
the values to the other value's type.
    ///
    ///      let x: Int8 = 21
    ///      let y: Int = 1000000
    ///      Int(x) + y          //
1000021
    ///
    /// - Parameters:
    ///   - lhs: The first value to add.
    ///   - rhs: The second value to add.
@inlinable public static func +(lhs:
EmptyAnimatableData, rhs:
EmptyAnimatableData) ->
EmptyAnimatableData
```

```
    /// Subtracts one value from another  
and produces their difference.  
    ///  
    /// The subtraction operator (`-`)  
calculates the difference of its two  
    /// arguments. For example:  
    ///  
    ///     8 - 3                      // 5  
    ///     -10 - 5                     //  
-15  
    ///     100 - -5                   //  
105  
    ///     10.5 - 100.0              //  
-89.5  
    ///  
    /// You cannot use `-` with arguments  
of different types. To subtract values  
    /// of different types, convert one  
of the values to the other value's type.  
    ///  
    ///     let x: UInt8 = 21  
    ///     let y: UInt = 1000000  
    ///     y - UInt(x)                //  
999979  
    ///  
    /// - Parameters:  
    ///   - lhs: A numeric value.  
    ///   - rhs: The value to subtract  
from `lhs`.  
    @inlinable public static func - (lhs:  
EmptyAnimatableData, rhs:  
EmptyAnimatableData) ->  
EmptyAnimatableData
```

```
    /// Multiplies each component of this
    /// value by the given value.
    @inlinable public mutating func
    scale(by rhs: Double)

    /// The dot-product of this
    /// animatable data instance with itself.
    @inlinable public var
    magnitudeSquared: Double { get }

    /// 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:
    EmptyAnimatableData, b:
    EmptyAnimatableData) -> Bool
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension EmptyAnimatableData : Sendable
{
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EmptyAnimatableData :  
BitwiseCopyable {  
}  
  
/// An empty, or identity, modifier, used  
during development to switch  
/// modifiers at compile time.  
///  
/// Use the empty modifier to switch  
modifiers at compile time during  
/// development. In the example below, in  
a debug build the ``Text``  
/// view inside `ContentView` has a  
yellow background and a red border.  
/// A non-debug build reflects the  
default system, or container supplied  
/// appearance.  
///  
///     struct EmphasizedLayout:  
ViewModifier {  
    ///         func body(content: Content)  
-> some View {  
        ///             content  
        ///                 .background(Color.yel  
low)  
        ///                 .border(Color.red)  
        ///             }  
        ///     }  
        ///  
///     struct ContentView: View {
```

```
///         var body: some View {
///             Text("Hello, World!")
///                 .modifier(modifier)
///         }
///
///         var modifier: some
ViewModifier {
///             #if DEBUG
///             return
EmphasizedLayout()
///             #else
///             return
EmptyModifier()
///             #endif
///         }
///
///     }
///
/// @available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct EmptyModifier : ViewModifier {
    public static let identity: EmptyModifier
        /// The type of view representing the
body.
    public typealias Body = Never
    @inlinable public init()
        /// Gets the current body of the
caller.
```

```
///  
/// `content` is a proxy for the view  
that will have the modifier  
/// represented by `Self` applied to  
it.  
@MainActor @preconcurrency public  
func body(content: EmptyModifier.Content)  
-> EmptyModifier.Body  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EmptyModifier : Sendable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EmptyModifier : BitwiseCopyable  
{  
}  
  
/// A view that doesn't contain any  
content.  
///  
/// You will rarely, if ever, need to  
create an `EmptyView` directly. Instead,  
/// `EmptyView` represents the absence of  
a view.  
///  
/// SwiftUI uses `EmptyView` in  
situations where a SwiftUI view type  
defines one  
/// or more child views with generic
```

```
parameters, and allows the child views to
/// be absent. When absent, the child
view's type in the generic type parameter
/// is `EmptyView`.  

///  

/// The following example creates an
indeterminate ``ProgressView`` without
/// a label. The ``ProgressView`` type
declares two generic parameters,
/// `Label` and `CurrentValueLabel`, for
the types used by its subviews.  

/// When both subviews are absent, like
they are here, the resulting type is
/// `ProgressView<EmptyView, EmptyView>`,
as indicated by the example's output:  

///  

///     let progressView = ProgressView()
///     print("\(type(of:progressView))")
///     // Prints:
ProgressView<EmptyView, EmptyView>  

///  

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct EmptyView : View {  

    /// Creates an empty view.
    @inlinable public init()  

        /// The type of view representing the
body of this view.
    ///  

    /// When you create a custom view,
Swift infers this type from your
```

```
    /// implementation of the required
    ``View/body-swift.property`` property.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Body = Never
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension EmptyView : Sendable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension EmptyView : BitwiseCopyable {
}

/// The base visual effect that you apply
additional effect to.
///
/// `EmptyVisualEffect` does not change
the appearance of the view
/// that it is applied to.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct EmptyVisualEffect : VisualEffect {

    /// Creates a new empty visual
effect.
    public init()

    /// The type defining the data to
```

```
animate.  
    @available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
    public typealias AnimatableData =  
EmptyAnimatableData  
}  
  
/// A selectability type that enables  
text selection by the person using your  
app.  
///  
/// Don't use this type directly.  
Instead, use  
``TextSelectability/enabled``.  
@available(iOS 15.0, macOS 12.0, *)  
@available(tvOS, unavailable)  
@available(watchOS, unavailable)  
public struct EnabledTextSelectability :  
TextSelectability {  
  
    /// A Boolean value that indicates  
whether the selectability type allows  
    /// selection.  
    ///  
    /// Conforming types, such as  
``EnabledTextSelectability`` and  
    /// ``DisabledTextSelectability``,  
return `true` or `false` for this  
    /// property as appropriate. SwiftUI  
expects this value for a given  
    /// selectability type to be  
constant, unaffected by global state.  
    public static let allowsSelection:
```

```
Bool
}

/// Creates an environment values,
transaction, container values,
/// or focused values entry.
///
/// ## Environment Values
///
/// Create ``EnvironmentValues`` entries
by extending the ``EnvironmentValues``
/// structure with new properties and
attaching the @Entry macro
/// to the variable declarations:
///
/// ````swift
/// extension EnvironmentValues {
///     @Entry var myCustomValue: String
///         = "Default value"
///     @Entry var anotherCustomValue =
true
/// }
/// ````

/// ## Transaction Values
///

/// Create ``Transaction`` entries by
extending the ``Transaction``
/// structure with new properties and
attaching the @Entry macro
/// to the variable declarations:
///
/// ````swift
```

```
/// extension Transaction {
///     @Entry var myCustomValue: String
= "Default value"
/// }
/// ``
/// 
/// ## Container Values
///
/// Create ``ContainerValues`` entries by
extending the ``ContainerValues``
/// structure with new properties and
attaching the @Entry macro
/// to the variable declarations:
///
/// ``swift
/// extension ContainerValues {
///     @Entry var myCustomValue: String
= "Default value"
/// }
/// ``
///
/// ## Focused Values
///
/// Since the default value for
``FocusedValues`` is always `nil`,
/// ``FocusedValues`` entries cannot
specify a different default value and
/// must have an Optional type.
///
/// Create ``FocusedValues`` entries by
extending the
/// ``FocusedValues`` structure with new
properties and attaching
```

```
/// the @Entry macro to the variable
declarations:
///
/// ````swift
/// extension FocusedValues {
///     @Entry var myCustomValue: String?
/// }
/// ````

@attached(accessor) @attached(peer,
names: prefixed(__Key__)) public macro
Entry() = #externalMacro(module:
"SwiftUIMacros", type: "EntryMacro")

/// A property wrapper that reads a value
from a view's environment.
///
/// Use the `Environment` property
wrapper to read a value
/// stored in a view's environment.
Indicate the value to read using an
/// ``EnvironmentValues`` key path in the
property declaration. For example, you
/// can create a property that reads the
color scheme of the current
/// view using the key path of the
``EnvironmentValues/colorScheme``
/// property:
///
///     @Environment(\.colorScheme) var
colorScheme: ColorScheme
///
/// You can condition a view's content on
the associated value, which
```

```
/// you read from the declared property's
``wrappedValue``. As with any property
/// wrapper, you access the wrapped value
by directly referring to the property:
///
///      if colorScheme == .dark { //
Checks the wrapped value.
///          DarkContent()
///      } else {
///          LightContent()
///      }
///
/// If the value changes, SwiftUI updates
any parts of your view that depend on
/// the value. For example, that might
happen in the above example if the user
/// changes the Appearance settings.
///
/// You can use this property wrapper to
read --- but not set --- an environment
/// value. SwiftUI updates some
environment values automatically based on
system
/// settings and provides reasonable
defaults for others. You can override
some
/// of these, as well as set custom
environment values that you define,
/// using the ``View/environment(_:_:)``
view modifier.
///
/// For the complete list of environment
values SwiftUI provides, see the
```

```
/// properties of the
``EnvironmentValues`` structure. For
information about
/// creating custom environment values,
see the ``Entry()`` macro.
///
/// Get an observable object
///
/// You can also use `Environment` to get
an observable object from a view's
/// environment. The observable object
must conform to the
///
<doc://com.apple.documentation/documentation/Observation/Observable>
/// protocol, and your app must set the
object in the environment using the
/// the object itself or a key path.
///
/// To set the object in the environment
using the object itself, use the
/// ``View/environment(_:)`` modifier:
///
///     @Observable
///     class Library {
///         var books: [Book] = [Book(),
Book(), Book()]
///
///         var availableBooksCount: Int
{
///
books.filter(\.isAvailable).count
/// }
```

```
    }
}
@main
struct BookReaderApp: App {
    @State private var library =
Library()
}

var body: some Scene {
    WindowGroup {
        LibraryView()
            .environment(libr
ary)
    }
}

}

/// To get the observable object using
/// its type, create a property and provide
/// the `Environment` property wrapper
/// the object's type:
///

struct LibraryView: View {
    @Environment(Library.self)
private var library

}

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

}

/// By default, reading an object from
/// the environment returns a non-optional
/// object when using the object type as
```

the key. This default behavior assumes  
/// that a view in the current hierarchy  
previously stored a non-optional  
/// instance of the type using the  
``View/environment(\_:)`` modifier. If  
/// a view attempts to retrieve an object  
using its type and that object isn't  
/// in the environment, SwiftUI throws an  
exception.

///

/// In cases where there is no guarantee  
that an object is in the environment,  
/// retrieve an optional version of the  
object as shown in the following code.  
/// If the object isn't available the  
environment, SwiftUI returns `nil`  
/// instead of throwing an exception.

///

///     @Environment(Library.self)  
private var library: Library?  
///

///     **### Get an observable object using a  
key path**

///

/// To set the object with a key path,  
use the ``View/environment(\_:\_:)``  
/// modifier:

///

///     @Observable  
///     class Library {  
///         var books: [Book] = [Book(),  
Book(), Book()]  
///

```
///         var availableBooksCount: Int
{
///         books.filter(\.isAvailable).count
///     }
/// }
/// @main
struct BookReaderApp: App {
    @State private var library =
Library()
///
///         var body: some Scene {
///             WindowGroup {
///                 LibraryView()
///                     .environment(\.li
brary, library)
///             }
///         }
/// }
///
/// To get the object, create a property
and specify the key path:
///
///         struct LibraryView: View {
///             @Environment(\.library)
private var library
///
///         var body: some View {
///             // ...
///         }
/// }
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen @propertyWrapper public struct  
Environment<Value> : DynamicProperty {  
  
    /// Creates an environment property  
    to read the specified key path.  
    ///  
    /// Don't call this initializer  
    directly. Instead, declare a property  
    /// with the ``Environment`` property  
    wrapper, and provide the key path of  
    /// the environment value that the  
    property should reflect:  
    ///  
    ///     struct MyView: View {  
    ///  
    @Environment(\.colorScheme) var  
    colorScheme: ColorScheme  
    ///  
    ///         // ...  
    ///     }  
    ///  
    /// SwiftUI automatically updates any  
    parts of `MyView` that depend on  
    /// the property when the associated  
    environment value changes.  
    /// You can't modify the environment  
    value using a property like this.  
    /// Instead, use the  
    ``View/environment(_:_:)`` view modifier  
    on a view to  
    /// set a value for a view hierarchy.
```

```
///  
/// - Parameter keyPath: A key path  
to a specific resulting value.  
@inlinable public init(_ keyPath:  
KeyPath<EnvironmentValues, Value>)  
  
    /// The current value of the  
environment property.  
    ///  
    /// The wrapped value property  
provides primary access to the value's  
data.  
    /// However, you don't access  
`wrappedValue` directly. Instead, you  
read the  
    /// property variable created with  
the ``Environment`` property wrapper:  
    ///  
    ///     @Environment(\.colorScheme)  
var colorScheme: ColorScheme  
    ///  
    ///     var body: some View {  
    ///         if colorScheme == .dark {  
    ///             DarkContent()  
    ///         } else {  
    ///             LightContent()  
    ///         }  
    ///     }  
    ///  
    ///     @inlinable public var wrappedValue:  
Value { get }  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Environment : Sendable where  
Value : Sendable {  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Environment {  
  
    /// Creates an environment property  
    to read an observable object from the  
    /// environment.  
    ///  
    /// - Important: This initializer  
    only accepts objects conforming to the  
    /// `Observable` protocol. For  
    reading environment objects that conform  
    to  
    /// `ObservableObject`, use  
    ``EnvironmentObject`` instead.  
    ///  
    /// Don't call this initializer  
    directly. Instead, declare a property  
    with  
    /// the ``Environment`` property  
    wrapper, passing the object's type to the  
    /// wrapper (using this syntax, the  
    object type can be omitted from the end  
    /// of property declaration):  
    ///  
    ///     @Observable final class  
Profile { ... }
```

```
///  
///     struct MyView: View {  
///  
@Environment(Profile.self) private var  
currentProfile  
///  
///         // ...  
///     }  
///  
/// - Warning: If no object has been  
set in the view's environment, this  
/// property will issue a fatal error  
when accessed. To safely check for the  
/// existence of an environment  
object, initialize the environment  
property  
    /// with an optional object type  
instead.  
///  
/// SwiftUI automatically updates any  
parts of `MyView` that depend on the  
/// property when the associated  
environment object changes.  
///  
/// You can't modify the environment  
object using a property like this.  
/// Instead, use the  
``View/environment(_:)`` view modifier on  
a view  
    /// to set an object for a view  
hierarchy.  
///  
/// - Parameter objectType: The type
```

```
of the `Observable` object to read
    /// from the environment.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public init(_ objectType: Value.Type)
where Value : AnyObject, Value :
Observable

    /// Creates an environment property
    to read an observable object from the
        /// environment, returning `nil` if
    no corresponding object has been set in
        /// the current view's environment.
    ///
    /// - Important: This initializer
    only accepts objects conforming to the
        /// `Observable` protocol. For
    reading environment objects that conform
    to
    /// `ObservableObject`, use
``EnvironmentObject`` instead.
    ///
    /// Don't call this initializer
    directly. Instead, declare an optional
        /// property with the ``Environment``
    property wrapper, passing the object's
        /// type to the wrapper:
    ///
    ///     @Observable final class
Profile { ... }
    ///
    ///     struct MyView: View {
    ///
```

```
@Environment(Profile.self) private var
currentProfile: Profile?
    /**
     /**
     *          // ...
     */
    /**
     * If no object has been set in the
view's environment, this property will
    /**
     * return `nil` as its wrapped
value.
    /**
     /**
     * SwiftUI automatically updates any
parts of `MyView` that depend on the
    /**
     * property when the associated
environment object changes.
    /**
     /**
     * You can't modify the environment
object using a property like this.
    /**
     * Instead, use the
``View/environment(_:)`` view modifier on
a view
    /**
     * to set an object for a view
hierarchy.
    /**
     /**
     * - Parameter objectType: The type
of the `Observable` object to read
    /**
     * from the environment.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public init<T>(_ objectType: T.Type)
where Value == T?, T : AnyObject, T :
Observable
}
```

```
/// A key for accessing values in the
environment.
///
/// You can create custom environment
values by extending the
/// ``EnvironmentValues`` structure with
new properties.
/// First declare a new environment key
type and specify a value for the
/// required ``defaultValue`` property:
///
///     private struct MyEnvironmentKey:
EnvironmentKey {
    static let defaultValue:
String = "Default value"
}

///
/// The Swift compiler automatically
infers the associated ``Value`` type as
the
/// type you specify for the default
value. Then use the key to define a new
/// environment value property:
///
///     extension EnvironmentValues {
    var myCustomValue: String {
        get
        { self[MyEnvironmentKey.self] }
        set
        { self[MyEnvironmentKey.self] =
newValue }
    }
}
```

```
///      }
///
/// Clients of your environment value
/// never use the key directly.
/// Instead, they use the key path of
/// your custom environment value property.
/// To set the environment value for a
/// view and all its subviews, add the
/// ``View/environment(_:_:)`` view
/// modifier to that view:
///
///     MyView()
///         .environment(\.myCustomValue,
/// "Another string")
///
/// As a convenience, you can also define
/// a dedicated view modifier to
/// apply this environment value:
///
///     extension View {
///         func myCustomValue(_
/// myCustomValue: String) -> some View {
///     environment(\.myCustomValue,
/// myCustomValue)
///         }
///     }
///
/// This improves clarity at the call
/// site:
///
///     MyView()
///         .myCustomValue("Another
```

```
string")
///
/// To read the value from inside
`MyView` or one of its descendants, use
the
/// ``Environment`` property wrapper:
///
///     struct MyView: View {
///         @Environment(\.myCustomValue)
var customValue: String
///
///         var body: some View {
///             Text(customValue) //
Displays "Another string".
///         }
///     }
///
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public protocol EnvironmentKey {

    /// The associated type representing
the type of the environment key's
    /// value.
associatedtype Value

    /// The default value for the
environment key.
    static var defaultValue: Self.Value {
get }
}

/// A property wrapper type for an
```

observable object that a parent or ancestor

/// view supplies.

///

/// An environment object invalidates the current view whenever the observable

/// object that conforms to

///

<doc://com.apple.documentation/documentation/Combine/ObservableObject>

/// changes. If you declare a property as an environment object, be sure

/// to set a corresponding model object on an ancestor view by calling its

/// ``View/environmentObject(\_:)`` modifier.

///

/// > Note: If your observable object conforms to the

///

<doc://com.apple.documentation/documentation/Observation/Observable>

/// protocol, use ``Environment`` instead of `EnvironmentObject` and set the

/// model object in an ancestor view by calling its ``View/environment(\_:)``

/// or ``View/environment(\_:\_:)`` modifiers.

`@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)`

`@MainActor @frozen @propertyWrapper`

`@preconcurrency public struct`

`EnvironmentObject<ObjectType> :`

```
DynamicProperty where ObjectType : ObservableObject {  
  
    /// A wrapper of the underlying  
    environment object that can create  
    bindings  
    /// to its properties using dynamic  
    member lookup.  
    @MainActor @dynamicMemberLookup  
    @frozen @preconcurrency public struct  
    Wrapper {  
  
        /// Returns a binding to the  
        resulting value of a given key path.  
        ///  
        /// - Parameter keyPath: A key  
        path to a specific resulting value.  
        ///  
        /// - Returns: A new binding.  
        @MainActor @preconcurrency public  
        subscript<Subject>(dynamicMember keyPath:  
        ReferenceWritableKeyPath<ObjectType,  
        Subject>) -> Binding<Subject> { get }  
    }  
  
    /// The underlying value referenced  
    by the environment object.  
    ///  
    /// This property provides primary  
    access to the value's data. However, you  
    /// don't access `wrappedValue`  
    directly. Instead, you use the property  
    /// variable created with the
```

```
``EnvironmentObject`` attribute.  
///  
/// When a mutable value changes, the  
new value is immediately available.  
/// However, a view displaying the  
value is updated asynchronously and may  
/// not show the new value  
immediately.  
@MainActor @inlinable @preconcurrency  
public var wrappedValue: ObjectType { get  
}  
  
    /// A projection of the environment  
object that creates bindings to its  
    /// properties using dynamic member  
lookup.  
    ///  
    /// Use the projected value to pass  
an environment object down a view  
    /// hierarchy.  
    @MainActor @preconcurrency public var  
projectedValue:  
EnvironmentObject<ObjectType>.Wrapper {  
get }  
  
    /// Creates an environment object.  
    @MainActor @preconcurrency public  
init()  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EnvironmentObject : Sendable {
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EnvironmentObject.Wrapper :  
Sendable {  
  
    /// A collection of environment values  
    // propagated through a view hierarchy.  
    ///  
    /// SwiftUI exposes a collection of  
    // values to your app's views in an  
    /// `EnvironmentValues` structure. To  
    // read a value from the structure,  
    /// declare a property using the  
    ``Environment`` property wrapper and  
    /// specify the value's key path. For  
    example, you can read the current locale:  
    ///  
    ///     @Environment(\.locale) var  
    locale: Locale  
    ///  
    /// Use the property you declare to  
    // dynamically control a view's layout.  
    /// SwiftUI automatically sets or updates  
    many environment values, like  
    /// ``EnvironmentValues/pixelLength``,  
    ``EnvironmentValues/scenePhase``, or  
    /// ``EnvironmentValues/locale``, based  
    on device characteristics, system state,  
    /// or user settings. For others, like  
    ``EnvironmentValues/lineLimit``, SwiftUI
```

```
/// provides a reasonable default value.  
///  
/// You can set or override some values  
using the ``View/environment(_:_:)``  
/// view modifier:  
///  
///     MyView()  
///         .environment(\.lineLimit, 2)  
///  
/// The value that you set affects the  
environment for the view that you modify  
/// --- including its descendants in the  
view hierarchy --- but only up to the  
/// point where you apply a different  
environment modifier.  
///  
/// SwiftUI provides dedicated view  
modifiers for setting some values, which  
/// typically makes your code easier to  
read. For example, rather than setting  
/// the ``EnvironmentValues/lineLimit``  
value directly, as in the previous  
/// example, you should instead use the  
``View/lineLimit(_:)`` modifier:  
///  
///     MyView()  
///         .lineLimit(2)  
///  
/// In some cases, using a dedicated view  
modifier provides additional  
/// functionality. For example, you must  
use the  
/// ``View/preferredColorScheme(_:)``
```

```
modifier rather than setting
/// ``EnvironmentValues/colorScheme``
directly to ensure that the new
/// value propagates up to the presenting
container when presenting a view
/// like a popover:
///
///     MyView()
///         .popover(isPresented:
$isPopped) {
///             PopoverContent()
///             .preferredColorScheme
(.dark)
///
/// Create a custom environment value by
declaring a new property
/// in an extension to the environment
values structure and applying
/// the ``Entry()`` macro to the variable
declaration:
///
///     extension EnvironmentValues {
///         @Entry var myCustomValue:
String = "Default value"
///     }
///
///     extension View {
///         func myCustomValue(_
myCustomValue: String) -> some View {
///         environment(\.myCustomValue,
myCustomValue)
```

```
    }
}
/// Clients of your value then access the
/// value in the usual way, reading it
/// with the ``Environment`` property
/// wrapper, and setting it with the
/// `myCustomValue` view modifier.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public struct EnvironmentValues :  
CustomStringConvertible {  
  
    /// Creates an environment values
instance.  
    /// You don't typically create an
instance of ``EnvironmentValues``
    /// directly. Doing so would provide
access only to default values that
    /// don't update based on system
settings or device characteristics.
    /// Instead, you rely on an
environment values' instance
    /// that SwiftUI manages for you when
you use the ``Environment``
    /// property wrapper and the
``View/environment(_:_:)`` view modifier.
    public init()  
  
    /// Accesses the environment value
associated with a custom key.
    ///
```

```
    /// Create a custom environment value
    // by declaring a new property
    /// in an extension to the
    environment values structure and applying
    /// the ``Entry()`` macro to the
    variable declaration:
    ///
    ///     extension EnvironmentValues {
    ///         @Entry var myCustomValue:
String = "Default value"
    ///     }
    ///
    /// You use custom environment values
the same way you use system-provided
    /// values, setting a value with the
``View/environment(_:_:)`` view
    /// modifier, and reading values with
the ``Environment`` property wrapper.
    /// You can also provide a dedicated
view modifier as a convenience for
    /// setting the value:
    ///
    ///     extension View {
    ///         func myCustomValue(_
myCustomValue: String) -> some View {
    ///
environment(\.myCustomValue,
myCustomValue)
    ///             }
    ///         }
    ///
public subscript<K>(key: K.Type) ->
K.Value where K : EnvironmentKey
```

```
    /// A string that represents the
contents of the environment values
    /// instance.
    public var description: String {
get }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension EnvironmentValues {

    /// The layout direction associated
with the current environment.
    ///
    /// Use this value to determine or
set whether the environment uses a
    /// left-to-right or right-to-left
direction.
    public var layoutDirection:
LayoutDirection
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension EnvironmentValues {

    /// A Boolean value that indicates
whether the view associated with this
    /// environment allows user
interaction.
    ///
    /// The default value is `true`.
```

```
    public var isEnabled: Bool
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension EnvironmentValues {

    /// An action that opens a URL.
    ///
    /// Read this environment value to
    get an ``OpenURLAction``
    /// instance for a given
    ``Environment``. Call the
    /// instance to open a URL. You call
    the instance directly because it
    /// defines a
    ``OpenURLAction/callAsFunction(_:)``
    method that Swift
    /// calls when you call the instance.
    ///
    /// For example, you can open a web
    site when the user taps a button:
    ///
    ///     struct OpenURLExample: View {
    ///         @Environment(\.openURL)
private var openURL
    ///
    ///         var body: some View {
    ///             Button {
    ///                 if let url =
URL(string: "https://www.example.com") {
    ///                     openURL(url)
    /// }
}
```

```
    } label: {
        Label("Get Help",
systemImage: "person.fill.questionmark")
    }
}

/// If you want to know whether the
action succeeds, add a completion
/// handler that takes a Boolean
value. In this case, Swift implicitly
/// calls the
``OpenURLAction/callAsFunction(_:completi
on:)`` method
/// instead. That method calls your
completion handler after it determines
/// whether it can open the URL, but
possibly before it finishes opening
/// the URL. You can add a handler to
the example above so that
/// it prints the outcome to the
console:
///
///     openURL(url) { accepted in
///         print(accepted ?
"Success" : "Failure")
///
/// The system provides a default
open URL action with behavior
/// that depends on the contents of
the URL. For example, the default
/// action opens a Universal Link in
```

```
the associated app if possible,  
    /// or in the user's default web  
browser if not.  
    ///  
    /// You can also set a custom action  
using the ``View/environment(_:_:)``  
    /// view modifier. Any views that  
read the action from the environment,  
    /// including the built-in ``Link``  
view and ``Text`` views with markdown  
    /// links, or links in attributed  
strings, use your action. Initialize an  
    /// action by calling the  
``OpenURLAction/init(handler:)``  
initializer with  
    /// a handler that takes a URL and  
returns an ``OpenURLAction/Result``:  
    ///  
    ///     Text("Visit [Example Company]  
(https://www.example.com) for details."  
    ///             .environment(\.openURL,  
OpenURLAction { url in  
    ///                     handleURL(url) //  
Define this method to take appropriate  
action.  
    ///                     return .handled  
    ///                 })  
    ///  
    /// SwiftUI translates the value that  
your custom action's handler  
    /// returns into an appropriate  
Boolean result for the action call.  
    /// For example, a view that uses the
```

```
action declared above
    /// receives `true` when calling the
action, because the
    /// handler always returns
``OpenURLAction/Result/handled``.
    public var openURL: OpenURLAction
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 8.0, *)
extension EnvironmentValues {

    /// A Boolean value that indicates
whether the display or environment
currently requires
    /// reduced luminance.
    ///
    /// When you detect this condition,
lower the overall brightness of your
view.
    /// For example, you can change
large, filled shapes to be stroked, and
choose
    /// less bright colors:
    ///
    ///
@Environment(\.isLuminanceReduced) var
isLuminanceReduced
    ///
    ///     var body: some View {
    ///         if isLuminanceReduced {
    ///             Circle()
    ///                 .stroke(Color.gra
```

```
y, lineWidth: 10)
    /**
     * } else {
     *     Circle()
     *         .fill(Color.white
)
    /**
     */
    /**
     */
    /**
     * In addition to the changes that
you make, the system could also
    /**
     * dim the display to achieve a
suitable brightness. By reacting to
    /**
     * `isLuminanceReduced`, you can
preserve contrast and readability
    /**
     * while helping to satisfy the
reduced brightness requirement.
    /**
    /**
     > Note: On watchOS, the system
typically sets this value to `true` when
the user
    /**
     lowers their wrist, but the
display remains on. Starting in watchOS
8, the system keeps your
    /**
     view visible on wrist down by
default. If you want the system to blur
the screen
    /**
     instead, as it did in earlier
versions of watchOS, set the value for
the
    /**
<doc://com.apple.documentation/documentation/BundleResources/Information_Property_List/WKSupportsAlwaysOnDisplay>
```

```
    /// key in your app's
    ///
<doc://com.apple.documentation/documentation/BundleResources/Information_Property_List>
    /// file to `false`.
    public var isLuminanceReduced: Bool
}

@available(iOS 14.0, macOS 11.0, tvOS 14.0, watchOS 7.0, *)
extension EnvironmentValues {

    /// The current redaction reasons applied to the view hierarchy.
    public var redactionReasons: RedactionReasons
}

@available(iOS 15.0, macOS 10.15, watchOS 9.0, *)
@available(tvOS, unavailable)
extension EnvironmentValues {

    /// The size to apply to controls within a view.
    ///
    /// The default is ``ControlSize/regular``.
    public var controlSize: ControlSize
}

@available(iOS 15.0, macOS 12.0, tvOS
```

```
15.0, watchOS 8.0, *)
extension EnvironmentValues {

    /// The material underneath the
    current view.
    ///
    /// This value is `nil` if the
    current background isn't one of the
    standard
    /// materials. If you set a material,
    the standard content styles enable
    /// their vibrant rendering modes.
    ///
    /// You set this value by calling one
    of the background modifiers that takes
    /// a ``ShapeStyle``, like
``View/background(_:ignoresSafeAreaEdges:
)``
    /// or
``View/background(_:in:fillStyle:)```, and
passing in a
    /// ``Material``. You can also set
    the value manually, using
    /// `nil` to disable vibrant
    rendering, or a ``Material`` instance to
    /// enable the vibrancy style
    associated with the specified material.
    public var backgroundMaterial:
Material?
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
```

```
extension EnvironmentValues {  
  
    /// The behavior materials should use  
    for their active state, defaulting to  
    /// `automatic`.  
    public var materialActiveAppearance:  
        MaterialActiveAppearance  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension EnvironmentValues {  
  
    /// The current symbol rendering  
    mode, or `nil` denoting that the  
    /// mode is picked automatically  
    using the current image and  
    /// foreground style as parameters.  
    public var symbolRenderingMode:  
        SymbolRenderingMode?  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EnvironmentValues {  
  
    /// An environment value that  
    indicates how a text view aligns its  
    lines  
    /// when the content wraps or  
    contains newlines.  
    ///  
    /// Set this value for a view
```

hierarchy by applying the  
    ///  
``View/multilineTextAlignment(\_:)`` view  
modifier. Views in the  
    /// hierarchy that display text, like  
``Text`` or ``TextEditor``, read the  
    /// value from the environment and  
adjust their text alignment accordingly.  
    ///  
    /// This value has no effect on a  
``Text`` view that contains only one  
    /// line of text, because a text view  
has a width that exactly matches the  
    /// width of its widest line. If you  
want to align an entire text view  
    /// rather than its contents, set the  
alignment of its container, like a  
    /// ``VStack`` or a frame that you  
create with the  
    ///  
``View/frame(minWidth:idealWidth:maxWidth  
:minHeight:idealHeight:maxHeight:alignmen  
t:)``  
    /// modifier.  
    ///  
    /// > Note: You can use this value to  
control the alignment of a ``Text``  
    /// view that you create with the  
``Text/init(\_:style:)`` initializer  
    /// to display localized dates and  
times, including when the view uses  
    /// only a single line, but only  
when that view appears in a widget.

```
public var multilineTextAlignment:  
TextAlignment
```

```
    /// A value that indicates how the  
layout truncates the last line of text to  
    /// fit into the available space.
```

```
    ///
```

```
    /// The default value is  
``Text/TruncationMode/tail``. Some  
controls,
```

```
    // however, might have a different  
default if appropriate.
```

```
public var truncationMode:  
Text.TruncationMode
```

```
    /// The distance in points between  
the bottom of one line fragment and the  
    /// top of the next.
```

```
    ///
```

```
    /// This value is always nonnegative.
```

```
public var lineSpacing: CGFloat
```

```
    /// A Boolean value that indicates  
whether inter-character spacing should  
    /// tighten to fit the text into the  
available space.
```

```
    ///
```

```
    /// The default value is `false`.
```

```
public var allowsTightening: Bool
```

```
    /// The minimum permissible  
proportion to shrink the font size to fit  
    /// the text into the available
```

space.

```
///
/// In the example below, a label
with a `minimumScaleFactor` of `0.5`
    /// draws its text in a font size as
small as half of the actual font if
    /// needed to fit into the space next
to the text input field:
///
///     HStack {
///         Text("This is a very long
label:")
///             .lineLimit(1)
///             .minimumScaleFactor(0
.5)
///         TextField("My Long Text
Field", text: $myTextField)
///             .frame(width: 250,
height: 50, alignment: .center)
///     }
///
/// ! [A screenshot showing the
effects of setting the minimumScaleFactor
on
    /// the text in a view](SwiftUI-
View-minimumScaleFactor.png)
///
/// You can set the minimum scale
factor to any value greater than `0` and
    /// less than or equal to `1`. The
default value is `1`.
///
/// SwiftUI uses this value to shrink
```

```
text that doesn't fit in a view when
    /// it's okay to shrink the text. For
example, a label with a
    /// `minimumScaleFactor` of `0.5`
draws its text in a font size as small as
    /// half the actual font if needed.
public var minimumScaleFactor:
CGFloat

    /// A stylistic override to transform
the case of `Text` when displayed,
    /// using the environment's locale.
    ///
    /// The default value is `nil`,
displaying the `Text` without any case
    /// changes.
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
public var textCase: Text.Case?
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension EnvironmentValues {

    /// The maximum number of lines that
text can occupy in a view.
    ///
    /// The maximum number of lines is
`1` if the value is less than `1`. If the
    /// value is `nil`, the text uses as
many lines as required. The default is
    /// `nil`.
```

```
    @available(iOS 13.0, macOS 10.15,
tvOS 13.0, watchOS 6.0, *)
public var lineLimit: Int?
}

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

    /// The prominence of the background
    underneath views associated with this
    /// environment.
    ///
    /// Foreground elements above an
    increased prominence background are
    /// typically adjusted to have higher
    contrast against a potentially vivid
    /// color, such as taking on a higher
    opacity monochrome appearance
    /// according to the `colorScheme`.
    System styles like `primary`,
    /// `secondary`, etc will
    automatically resolve to an appropriate
    style in
    /// this context. The property can be
    read and used for custom styled
    /// elements.
    ///
    /// In the example below, a custom
    star rating element is in a list row
    /// alongside some text. When the row
    is selected and has an increased
    /// prominence appearance, the text
```

```
and star rating will update their
    /// appearance, the star rating
replacing its use of yellow with the
    /// standard `secondary` style.
    ///
    ///     struct RecipeList: View {
    ///         var recipes: [Recipe]
    ///         @Binding var
selectedRecipe: Recipe.ID?
    ///
    ///         var body: some View {
    ///             List(recipes,
selection: $selectedRecipe) {
    ///
RecipeListRow(recipe: $0)
    ///                     }
    ///                     }
    ///                     }
    ///
    ///     struct RecipeListRow: View {
    ///         var recipe: Recipe
    ///         var body: some View {
    ///             VStack(alignment: .leading) {
    ///                 HStack(alignment:
.firstTextBaseline) {
    ///
Text(recipe.name)
    ///                     Spacer()
    ///
StarRating(rating: recipe.rating)
    ///                     }
    ///
```



```
    /**
     */
    /**
     * Note that the use of
     `backgroundProminence` was used by a view
     that
     /**
      * was nested in additional stack
      containers within the row. This ensured
      /**
       * that the value correctly
       reflected the environment within the list
       row
       /**
        * itself, as opposed to the
        environment of the list as a whole. One
        way
        /**
         * to ensure correct resolution
         would be to prefer using this in a custom
         /**
          * ShapeStyle instead, for example:
          /**
          /**
           * private struct StarRating:
View {
    /**
     * var rating: Int
    /**
    /**
     * var body: some View {
    /**
             HStack(spacing: 1) {
    /**
    /**
     * ForEach(0..<rating, id: \.self) { _ in
    /**
     * Image(systemName: "star.fill")
    /**
                     }
    /**
                     }
    /**
     * .foregroundStyle(Fill
Style())
    /**
                     .imageScale(.small)
    /**
}
```

```
    /**
     */
    /**
     * extension StarRating {
     *     struct FillStyle:
ShapeStyle {
     *         func resolve(in env: EnvironmentValues) -> some ShapeStyle {
     *             switch env.backgroundProminence {
     *                 /**
     *                  case .increased:
return AnyShapeStyle(.secondary)
     *                 /**
     *                  default: return AnyShapeStyle(.yellow)
     *             }
     *         }
     *     }
     * }
     */
     /**
     * Views like `List` and `Table` as well as standard shape styles like
     * `ShapeStyle.selection` will automatically update the background
     * prominence of foreground views.
For custom backgrounds, this environment
     * property can be explicitly set on views above custom backgrounds.
public var backgroundProminence: BackgroundProminence
}

@available(iOS 16.0, macOS 13.0, tvOS 16.0, watchOS 9.0, *)
extension EnvironmentValues {
```

```
    /// An optional style that overrides
    the default system background
    /// style when set.
    public var backgroundStyle:
AnyShapeStyle?
}

@available(iOS 18.0, macOS 10.15, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension EnvironmentValues {

    /// Whether views and styles in this
    environment should prefer an active
    /// appearance over an inactive
    appearance.
    ///
    /// On macOS, views in the focused
    window (also referred to as the "key"
    /// window) should appear active.
    Some contexts also appear active in other
    /// circumstances, such as the
    contents of a window toolbar appearing
    active
    /// when the window is not focused
    but is the main window.
    ///
    /// Typical adjustments made when a
    view does not appear active include:
    /// - Uses of `Color.accentColor`
    should generally be removed or replaced
    /// with a desaturated style.
    /// - Text and image content in
```

```
sidebars should appear dimmer.  
    /// - Buttons with destructive  
actions should appear disabled.  
    /// - `ShapeStyle.selection` and  
selection in list and tables will  
    /// automatically become a grey  
color  
    ///  
    /// Custom views, styles, and shape  
styles can use this to adjust their  
    /// own appearance:  
    ///  
    ///     struct  
ProminentPillButtonStyle: ButtonStyle {  
    ///  
    @Environment(\.appearsActive) private var  
appearsActive  
    ///  
    ///         func  
makeBody(configuration: Configuration) ->  
some View {  
    ///             configuration.label  
    ///                 .lineLimit(1)  
    ///                 .padding(.horizon  
tal, 8)  
    ///                 .padding(.vertica  
l, 2)  
    ///                 .frame(minHeight:  
20)  
    ///                     .overlay(Capsule()  
).strokeBorder(.tertiary))  
    ///                     .background(appe  
rsActive ? Color.accentColor : .clear,
```

```
in: .capsule)
    /**
     * contentShape(.ca
psule)
    /**
     */
    /**
     */
    /**
     * On all other platforms, this
value is always `true`.
    /**
     * This is bridged with
`UITraitCollection.activeAppearance` for
UIKit
    /**
     * hosted content.
    @available(iOS 18.0, macOS 10.15,
tvOS 18.0, watchOS 11.0, visionOS 2.0, *)
    @backDeployed(before: macOS 15.0)
    public var appearsActive: Bool
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension EnvironmentValues {

    /**
     * The current Dynamic Type size.
    /**
     * This value changes as the user's
chosen Dynamic Type size changes. The
    /**
     * default value is device-
dependent.
    /**
     * When limiting the Dynamic Type
size, consider if adding a
    /**
     * large content view with
```

```
``View/accessibilityShowsLargeContentView
er()``
    /// would be appropriate.
    ///
    /// On macOS, this value cannot be
    changed by users and does not affect the
    /// text size.
    public var dynamicTypeSize:
DynamicTypeSize
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, visionOS 1.0, *)
extension EnvironmentValues {

    /// The preferred size of the
    content.
    ///
    /// The default value is
``ContentSizeCategory/large``.
    @available(iOS, introduced: 13.0,
deprecated: 100000.0, renamed:
"dynamicTypeSize")
    @available(macOS, introduced: 10.15,
deprecated: 100000.0, renamed:
"dynamicTypeSize")
    @available(tvOS, introduced: 13.0,
deprecated: 100000.0, renamed:
"dynamicTypeSize")
    @available(watchOS, introduced: 6.0,
deprecated: 100000.0, renamed:
"dynamicTypeSize")
    @available(visionOS, introduced: 1.0,
```

```
deprecated: 100000.0, renamed:  
"dynamicTypeSize")  
    public var sizeCategory:  
ContentSizeCategory  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension EnvironmentValues {  
  
    /// Reads an observable object of the  
    /// specified type from the environment.  
    ///  
    /// - Important: This subscript only  
    /// supports reading objects that conform  
    /// to the `Observable` protocol.  
    ///  
    /// Use this subscript to read the  
    /// environment object of a specific type  
    /// from an instance of  
    /// ``EnvironmentValues``, such as when  
    /// accessing the  
    /// ``GraphicsContext/environment``  
    /// property of a graphics context:  
    ///  
    ///     @Observable final class  
Profile { ... }  
    ///  
    ///     Canvas { context, size in  
    ///             let currentProfile =  
context.environment[Profile.self]  
    ///             ...  
    /// }
```

```
    /**
     * - Parameter objectType: The type
     * of the `Observable` object to read
     *      from the environment.
     */
    /**
     * - Returns: The environment object
     * of the specified type, or `nil` if no
     *      object of that type has been
     * set in this environment.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public subscript<T>(objectType:
T.Type) -> T? where T : AnyObject, T :
Observable
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension EnvironmentValues {

    /// The prominence to apply to
    section headers within a view.
    /**
     * The default is
     * ``Prominence/standard``.
    public var headerProminence:
Prominence
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, *)
@available(watchOS, unavailable)
extension EnvironmentValues {
```

```
    /// The allowed dynamic range for the
view, or nil.
    public var allowedDynamicRange:
Image.DynamicRange?
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension EnvironmentValues {

    /// The symbol variant to use in this
environment.
    ///
    /// You set this environment value
indirectly by using the
    /// ``View/symbolVariant(_:)`` view
modifier. However, you access the
    /// environment variable directly
using the ``View/environment(_:_:)``
    /// modifier. Do this when you want
to use the ``SymbolVariants/none``
    /// variant to ignore the value
that's already in the environment:
    ///
    ///     HStack {
    ///         Image(systemName:
"heart")
    ///         Image(systemName:
"heart")
    ///             .environment(\.symbol
Variants, .none)
    ///     }
}
```

```
    ///      .symbolVariant(.fill)
    ///
    /// ! [A screenshot of two heart
symbols. The first is filled while the
    /// second is outlined.]
(SymbolVariants-none-1)
    public var symbolVariants:
SymbolVariants
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension EnvironmentValues {

    /// The current method of animating
the contents of views.
    public var contentTransition:
ContentTransition

    /// A Boolean value that controls
whether views that render content
    /// transitions use GPU-accelerated
rendering.
    ///
    /// Setting this value to `true`
causes SwiftUI to wrap content
transitions
    /// with a
``View/drawingGroup(opaque:colorMode:)``
modifier.
    public var
contentTransitionAddsDrawingGroup: Bool
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EnvironmentValues {  
  
    /// The default font of this  
    environment.  
    public var font: Font?  
  
    /// The image scale for this  
    environment.  
    @available(macOS 11.0, *)  
    public var imageScale: Image.Scale  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EnvironmentValues {  
  
    /// The display scale of this  
    environment.  
    public var displayScale: CGFloat  
  
    /// The size of a pixel on the  
    screen.  
    ///  
    /// This value is usually equal to  
    `1` divided by  
    ///  
    ``EnvironmentValues/displayScale``.  
    public var pixelLength: CGFloat { get  
}  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EnvironmentValues {  
  
    /// The font weight to apply to text.  
    ///  
    /// This value reflects the value of  
    /// the Bold Text display setting found in  
    /// the Accessibility settings.  
    public var legibilityWeight:  
        LegibilityWeight?  
  
    /// The current locale that views  
    /// should use.  
    public var locale: Locale  
  
    /// The current calendar that views  
    /// should use when handling dates.  
    public var calendar: Calendar  
  
    /// The current time zone that views  
    /// should use when handling dates.  
    public var timeZone: TimeZone  
  
    /// The active appearance expected of  
    /// controls in a window.  
    ///  
    /// `ControlActiveState` and  
    /// `EnvironmentValues.controlActiveState`  
    /// are  
    /// deprecated, use  
    /// `EnvironmentValues.appearsActive`
```

instead.

```
///  
/// Starting with macOS 15.0, the  
value of this environment property is  
/// strictly mapped to and from  
`EnvironmentValues.appearsActive` as  
follows:  
    /// - `appearsActive == true`,  
`controlActiveState` returns `.key`  
    /// - `appearsActive == false`,  
`controlActiveState` returns `.inactive`  
    /// - `controlActiveState` is set to  
.key` or `.active`, `appearsActive`  
    /// will be set to `true`.  
    /// - `controlActiveState` is set to  
.inactive`, `appearsActive` will be  
    /// set to `false`.  
    @available(iOS, unavailable)  
    @available(macOS, introduced: 10.15,  
deprecated: 100000.0, message: "Use  
instead.")  
    @available(tvOS, unavailable)  
    @available(watchOS, unavailable)  
    @available(visionOS, unavailable)  
    public var controlActiveState:  
ControlActiveState  
  
    /// The horizontal size class of this  
environment.  
    ///  
    /// You receive a  
``UserInterfaceSizeClass`` value when you
```

read this

    /// environment value. The value  
    tells you about the amount of horizontal

    /// space available to the view that  
    reads it. You can read this

    /// size class like any other of the  
    ``EnvironmentValues``, by creating a

    /// property with the ``Environment``  
    property wrapper:

    ///

    ///

```
@Environment(\.horizontalSizeClass)
```

```
private var horizontalSizeClass
```

    ///

    /// SwiftUI sets this size class  
    based on several factors, including:

    ///

    /// \* The current device type.

    /// \* The orientation of the device.

    /// \* The appearance of Slide Over

and Split View on iPad.

    ///

    /// Several built-in views change  
    their behavior based on this size class.

    /// For example, a ``NavigationView``  
    presents a multicolumn view when

    /// the horizontal size class is

    ``UserInterfaceSizeClass/regular``,

    /// but a single column otherwise.

You can also adjust the appearance of

    /// custom views by reading the size  
    class and conditioning your views.

    /// If you do, be prepared to handle

```
size class changes while
    /// your app runs, because factors
like device orientation can change at
    /// runtime.
    ///
    /// In watchOS, the horizontal size
class is always
    ///
``UserInterfaceSizeClass/compact``. In
macOS, and tvOS, it's always
    ///
``UserInterfaceSizeClass/regular``.
    ///
    /// Writing to the horizontal size
class in the environment
    /// before macOS 14.0, tvOS 17.0, and
watchOS 10.0 is not supported.
    @available(iOS 13.0, macOS 10.15,
tvOS 13.0, watchOS 6.0, *)
    @backDeployed(before: macOS 14.0,
tvOS 17.0, watchOS 10.0)
    public var horizontalSizeClass:
UserInterfaceSizeClass?
```

  

```
    /// The vertical size class of this
environment.
    ///
    /// You receive a
``UserInterfaceSizeClass`` value when you
read this
    /// environment value. The value
tells you about the amount of vertical
    /// space available to the view that
```

reads it. You can read this

```
    /// size class like any other of the
``EnvironmentValues``, by creating a
    /// property with the ``Environment``
property wrapper:
```

```
    ///
    ///
```

```
@Environment(\.verticalSizeClass) private
var verticalSizeClass
```

```
    ///
    ///
```

/// SwiftUI sets this size class  
based on several factors, including:

```
    ///
    ///
```

/// \* The current device type.

/// \* The orientation of the device.

```
    ///
    ///
```

/// You can adjust the appearance of  
custom views by reading this size

/// class and conditioning your  
views. If you do, be prepared to

/// handle size class changes while  
your app runs, because factors like

/// device orientation can change at  
runtime.

```
    ///
    ///
```

/// In watchOS, the vertical size  
class is always

```
    ///
    ///
```

``UserInterfaceSizeClass/compact``. In  
macOS, and tvOS, it's always

```
    ///
    ///
```

``UserInterfaceSizeClass/regular``.

```
    ///
    ///
```

```
    /// Writing to the vertical size
    class in the environment
        /// before macOS 14.0, tvOS 17.0, and
        watchOS 10.0 is not supported.
        @available(iOS 13.0, macOS 10.15,
tvOS 13.0, watchOS 6.0, *)
        @backDeployed(before: macOS 14.0,
tvOS 17.0, watchOS 10.0)
        public var verticalSizeClass:
UserInterfaceSizeClass?
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension EnvironmentValues {

    /// A Boolean value that indicates
    whether the user has enabled an assistive
    /// technology.
    public var accessibilityEnabled: Bool
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension EnvironmentValues {

    /// Whether the system preference for
    Differentiate without Color is enabled.
    ///
    /// If this is true, UI should not
    convey information using color alone
    /// and instead should use shapes or
    glyphs to convey information.
```

```
public var
accessibilityDifferentiateWithoutColor:
Bool { get }

    /// Whether the system preference for
Reduce Transparency is enabled.
    ///
    /// If this property's value is true,
UI (mainly window) backgrounds should
        /// not be semi-transparent; they
should be opaque.

public var
accessibilityReduceTransparency: Bool {
get }

    /// Whether the system preference for
Reduce Motion is enabled.
    ///
    /// If this property's value is true,
UI should avoid large animations,
        /// especially those that simulate
the third dimension.

public var accessibilityReduceMotion:
Bool { get }

    /// Whether the system preference for
Invert Colors is enabled.
    ///
    /// If this property's value is true
then the display will be inverted.
        /// In these cases it may be needed
for UI drawing to be adjusted to in
        /// order to display optimally when
```

inverted.

```
    public var accessibilityInvertColors:  
Bool { get }  
}
```

```
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
extension EnvironmentValues {
```

/// Whether the system preference for  
Show Button Shapes is enabled.

```
    ///
```

/// If this property's value is true,  
interactive custom controls

/// such as buttons should be drawn  
in such a way that their edges

/// and borders are clearly visible.

```
    public var  
accessibilityShowButtonShapes: Bool { get  
}  
}
```

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

/// Whether the setting to reduce  
flashing or strobing lights in video

/// content is on. This setting can  
also be used to determine if UI in

/// playback controls should be shown  
to indicate upcoming content that

/// includes flashing or strobing

```
lights.  
    public var  
accessibilityDimFlashingLights: Bool {  
get }  
  
    /// Whether the setting for playing  
animations in an animated image is  
    /// on. When this value is false, any  
presented image that contains  
    /// animation should not play  
automatically.  
    public var  
accessibilityPlayAnimatedImages: Bool {  
get }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension EnvironmentValues {  
  
    /// The color scheme of this  
environment.  
    ///  
    /// Read this environment value from  
within a view to find out if SwiftUI  
    /// is currently displaying the view  
using the ``ColorScheme/light`` or  
    /// ``ColorScheme/dark`` appearance.  
The value that you receive depends on  
    /// whether the user has enabled Dark  
Mode, possibly superseded by  
    /// the configuration of the current  
presentation's view hierarchy.
```

```
///  
///      @Environment(\.colorScheme)  
private var colorScheme  
///  
///      var body: some View {  
///          Text(colorScheme == .dark  
? "Dark" : "Light")  
///      }  
///  
/// You can set the `colorScheme`  
environment value directly,  
/// but that usually isn't what you  
want. Doing so changes the color  
/// scheme of the given view and its  
child views but *not* the views  
/// above it in the view hierarchy.  
Instead, set a color scheme using the  
/// ``View/preferredColorScheme(_:)``  
modifier, which also propagates the  
/// value up through the view  
hierarchy to the enclosing presentation,  
like  
/// a sheet or a window.  
///  
/// When adjusting your app's user  
interface to match the color scheme,  
/// consider also checking the  
``EnvironmentValues/colorSchemeContrast``  
/// property, which reflects a  
system-wide contrast setting that the  
user  
/// controls. For information, see  
///
```

```
<doc://com.apple.documentation/design/human-interface-guidelines/accessibility#Color-and-effects>
    /// in the Human Interface Guidelines.
    /**
     /// > Note: If you only need to provide different colors or
     /// images for different color scheme and contrast settings, do that in
     /// your app's Asset Catalog. See
     /**
<doc://com.apple.documentation/documentation/Xcode/Asset-Management>.
public var colorScheme: ColorScheme

    /// The contrast associated with the color scheme of this environment.
    /**
     /// Read this environment value from within a view to find out if SwiftUI
     /// is currently displaying the view using ``ColorSchemeContrast/standard``
     /// or ``ColorSchemeContrast/increased`` contrast. The value that you read
     /// depends entirely on user settings, and you can't change it.
    /**
    /**
@Environment(\.colorSchemeContrast)
private var colorSchemeContrast
    /**
```

```
    ///      var body: some View {
    ///          Text(colorSchemeContrast
== .standard ? "Standard" : "Increased")
    ///      }
    ///
    /// When adjusting your app's user
interface to match the contrast,
    /// consider also checking the
``EnvironmentValues/colorScheme``
property
    /// to find out if SwiftUI is
displaying the view with a light or dark
    /// appearance. For information, see
    ///
<doc://com.apple.documentation/design/hum
an-interface-guidelines/accessibility#Col
or-and-effects>
    /// in the Human Interface
Guidelines.
    ///
    /// > Note: If you only need to
provide different colors or
    /// images for different color scheme
and contrast settings, do that in
    /// your app's Asset Catalog. See
    ///
<doc://com.apple.documentation/documentation/Xcode/Asset-Management>.
public var colorSchemeContrast:
ColorSchemeContrast { get }
}

/// A modifier that must resolve to a
```

```
concrete modifier in an environment  
before  
/// use.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
public protocol EnvironmentalModifier :  
ViewModifier where Self.Body == Never {  
  
    /// The type of modifier to use after  
    being resolved.  
    associatedtype ResolvedModifier :  
ViewModifier  
  
    /// Resolve to a concrete modifier in  
    the given `environment`.  
    func resolve(in environment:  
EnvironmentValues) ->  
Self.ResolvedModifier  
}  
  
/// A set of key modifiers that you can  
add to a gesture.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct EventModifiers :  
OptionSet {  
  
    /// The raw value.  
    public let rawValue: Int  
  
    /// Creates a new set from a raw  
    value.  
    ///
```

```
    /// - Parameter rawValue: The raw  
    value with which to create the key  
    /// modifier.  
    public init(rawValue: Int)  
  
    /// The Caps Lock key.  
    public static let capsLock:  
EventModifiers  
  
    /// The Shift key.  
    public static let shift:  
EventModifiers  
  
    /// The Control key.  
    public static let control:  
EventModifiers  
  
    /// The Option key.  
    public static let option:  
EventModifiers  
  
    /// The Command key.  
    public static let command:  
EventModifiers  
  
    /// Any key on the numeric keypad.  
    public static let numericPad:  
EventModifiers  
  
    /// The Function key.  
    @available(iOS, deprecated: 15.0,  
message: "Function modifier is reserved  
for system applications")
```

```
    @available(macOS, deprecated: 12.0,
message: "Function modifier is reserved
for system applications")
    @available(tvOS, deprecated: 15.0,
message: "Function modifier is reserved
for system applications")
    @available(watchOS, deprecated: 8.0,
message: "Function modifier is reserved
for system applications")
    @available(visionOS, deprecated: 1.0,
message: "Function modifier is reserved
for system applications")
    public static let function:
EventModifiers

    /// All possible modifier keys.
public static let all: EventModifiers

    /// The type of the elements of an
array literal.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias ArrayLiteralElement
= EventModifiers

    /// The element type of the option
set.
    ///
    /// To inherit all the default
implementations from the `OptionSet`-
protocol,
    /// the `Element` type must be
`Self`, the default.
```

```
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Element =
EventModifiers

        /// The raw type that can be used to
represent all values of the conforming
        /// type.
        ///
        /// Every distinct value of the
conforming type has a corresponding
unique
        /// value of the `RawValue` type, but
there may be values of the `RawValue`'
        /// type that don't have a
corresponding value of the conforming
type.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias RawValue = Int
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension EventModifiers : Sendable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension EventModifiers :
BitwiseCopyable {
}
```

```
/// A schedule for updating a timeline
view at the start of every minute.
///
/// You can also use
``TimelineSchedule/everyMinute`` to
construct this
/// schedule.
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
public struct EveryMinuteTimelineSchedule
: TimelineSchedule, Sendable {

    /// The sequence of dates in an every
    minute schedule.
    ///
    /// The
    ``EveryMinuteTimelineSchedule/entries(fro
m:mode:)`` method returns
        /// a value of this type, which is a
        ///
<doc://com.apple.documentation/documentat
ion/Swift/Sequence>
    /// of dates, one per minute, in
    ascending order. A ``TimelineView`` that
        /// you create updates its content at
    the moments in time corresponding to
        /// the dates included in the
    sequence.
    public struct Entries : Sequence,
IteratorProtocol, Sendable {

        /// Advances to the next element
        and returns it, or `nil` if no next
```

```
element
    /// exists.
    ///
    /// Repeatedly calling this
method returns, in order, all the
elements of the
    /// underlying sequence. As soon
as the sequence has run out of elements,
all
        /// subsequent calls return
`nil`.
    ///
    /// You must not call this method
if any other copy of this iterator has
been
        /// advanced with a call to its
`next()` method.
    ///
    /// The following example shows
how an iterator can be used explicitly to
        /// emulate a `for`-`in` loop.
First, retrieve a sequence's iterator,
and
        /// then call the iterator's
`next()` method until it returns `nil`.
    ///
    ///     let numbers = [2, 3, 5,
7]
        ///     var numbersIterator =
numbers.makeIterator()
    ///
    ///     while let num =
numbersIterator.next() {
```

```
    ///         print(num)
    ///     }
    ///     // Prints "2"
    ///     // Prints "3"
    ///     // Prints "5"
    ///     // Prints "7"
    ///
    /// - Returns: The next element
    in the underlying sequence, if a next
    element
    /// exists; otherwise, `nil`.
public mutating func next() ->
Date?
```

```
    /// A type representing the
sequence's elements.
@available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
public typealias Element = Date

    /// A type that provides the
sequence's iteration interface and
    /// encapsulates its iteration
state.
@available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
public typealias Iterator =
EveryMinuteTimelineSchedule.Entries
}

    /// Creates a per-minute update
schedule.
///
```

```
    /// Use the
    ``EveryMinuteTimelineSchedule/entries(fro
m:mode:)`` method
    /// to get the sequence of dates.
public init()

    /// Provides a sequence of per-minute
dates starting from a given date.
    ///
    /// A ``TimelineView`` that you
create with an every minute schedule
    /// calls this method to ask the
schedule when to update its content.
    /// The method returns a sequence of
per-minute dates in increasing
    /// order, from earliest to latest,
that represents
    /// when the timeline view updates.
    ///
    /// For a `startDate` that's exactly
minute-aligned, the
    /// schedule's sequence of dates
starts at that time. Otherwise, it
    /// starts at the beginning of the
specified minute. For
    /// example, for start dates of both
`10:09:32` and `10:09:00`, the first
    /// entry in the sequence is
`10:09:00`.

    ///
    /// - Parameters:
    ///   - startDate: The date from
which the sequence begins.
```

```
    /// - mode: The mode for the update
    schedule.

    /// - Returns: A sequence of per-
    minute dates in ascending order.
    public func entries(from startDate:
Date, mode: TimelineScheduleMode) ->
EveryMinuteTimelineSchedule.Entries
}

/// A gesture that consists of two
gestures where only one of them can
succeed.
///
/// The `ExclusiveGesture` gives
precedence to its first gesture.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct
ExclusiveGesture<First, Second> : Gesture
where First : Gesture, Second : Gesture {

    /// The value of an exclusive gesture
    that indicates which of two gestures
    /// succeeded.
    @frozen public enum Value {

        /// The first of two gestures
        succeeded.
        case first(First.Value)

        /// The second of two gestures
        succeeded.
        case second(Second.Value)
    }
}
```

```
}

/// The first of two gestures.
public var first: First

/// The second of two gestures.
public var second: Second

/// Creates a gesture from two
gestures where only one of them succeeds.
///
/// - Parameters:
///   - first: The first of two
gestures. This gesture has precedence
over
///       the other gesture.
///   - second: The second of two
gestures.
@inlinable public init(_ first:
First, _ second: Second)

/// The type of gesture representing
the body of `Self`.
@available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
public typealias Body = Never
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ExclusiveGesture.Value :
Sendable where First.Value : Sendable,
Second.Value : Sendable {
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ExclusiveGesture.Value :  
Equatable where First.Value : Equatable,  
Second.Value : Equatable {
```

```
    /// 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:  
ExclusiveGesture<First, Second>.Value, b:  
ExclusiveGesture<First, Second>.Value) ->  
Bool
```

```
}
```

```
/// A schedule for updating a timeline  
view at explicit points in time.
```

```
///
```

```
/// You can also use  
``TimelineSchedule/explicit(_:)`` to  
construct this  
/// schedule.
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
public struct  
ExplicitTimelineSchedule<Entries> :  
TimelineSchedule where Entries :  
Sequence, Entries.Element == Date {  
  
    /// Creates a schedule composed of an  
    explicit sequence of dates.  
    ///  
    /// Use the  
``ExplicitTimelineSchedule/entries(from:m  
ode:)`` method  
    /// to get the sequence of dates.  
    ///  
    /// - Parameter dates: The sequence  
    of dates at which a timeline view  
    /// updates. Use a monotonically  
    increasing sequence of dates,  
    /// and ensure that at least one is  
    in the future.  
    public init(_ dates: Entries)  
  
    /// Provides the sequence of dates  
    with which you initialized the schedule.  
    ///  
    /// A ``TimelineView`` that you  
    create with a schedule calls this  
    /// ``TimelineSchedule`` method to  
    ask the schedule when to update its  
    /// content. The explicit timeline  
    schedule implementation  
    /// of this method returns the
```

```
unmodified sequence of dates that you
    /// provided when you created the
schedule with
    ///
``TimelineSchedule/explicit(_:)``. As a
result, this particular
    /// implementation ignores the
`startDate` and `mode` parameters.
    ///
    /// - Parameters:
    ///   - startDate: The date from
which the sequence begins. This
    ///   particular implementation of
the protocol method ignores the start
    ///   date.
    ///   - mode: The mode for the update
schedule. This particular
    ///   implementation of the
protocol method ignores the mode.
    /// - Returns: The sequence of dates
that you provided at initialization.
    public func entries(from startDate:
Date, mode: TimelineScheduleMode) ->
Entries
}

/// A shape provider that fills its
shape.
///
/// You do not create this type directly,
it is the return type of `Shape.fill`.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
```

```
@frozen public struct
FillShapeView<Content, Style, Background>
: ShapeView where Content : Shape,
Style : ShapeStyle, Background : View {

    /// The shape that this type draws
and provides for other drawing
    /// operations.
    public var shape: Content

    /// The style that fills this view's
shape.
    public var style: Style

    /// The fill style used when filling
this view's shape.
    public var fillStyle: FillStyle

    /// The background shown beneath this
view.
    public var background: Background

    /// Create a FillShapeView.
    public init(shape: Content, style:
Style, fillStyle: FillStyle, background:
Background)

    /// The type of view representing the
body of this view.
    ///
    /// When you create a custom view,
Swift infers this type from your
    /// implementation of the required
```

```
``View/body-swift.property`` property.
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Body = Never
}

/// A style for rasterizing vector
shapes.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct FillStyle : Equatable {

    /// A Boolean value that indicates
whether to use the even-odd rule when
    /// rendering a shape.
    ///
    /// When `isEOFilled` is `false`, the
style uses the non-zero winding number
    /// rule.
    public var isEOFilled: Bool

    /// A Boolean value that indicates
whether to apply antialiasing to the
    /// edges of a shape.
    public var isAntialiased: Bool

    /// Creates a new fill style with the
specified settings.
    ///
    /// - Parameters:
    ///   - eoFill: A Boolean value that
indicates whether to use the even-odd
```

```
    /// rule for rendering a shape.  
    Pass `false` to use the non-zero winding  
    /// number rule instead.  
    /// - antialiased: A Boolean value  
    that indicates whether to use  
    /// antialiasing when rendering  
    the edges of a shape.  
    @inlinable public init(eoFill: Bool =  
    false, antialiased: Bool = true)  
  
        /// 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: FillStyle,  
        b: FillStyle) -> Bool  
    }  
  
    @available(iOS 13.0, macOS 10.15, tvOS  
    13.0, watchOS 6.0, *)  
    extension FillStyle : Sendable {  
    }  
  
    @available(iOS 13.0, macOS 10.15, tvOS  
    13.0, watchOS 6.0, *)
```

```
extension FillStyle : BitwiseCopyable {  
}  
  
/// The default text variant preference  
that chooses the largest available  
/// variant.  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
public struct FixedTextVariant :  
TextVariantPreference, Sendable {  
}  
  
/// An environment-dependent font.  
///  
/// The system resolves a font's value at  
the time it uses the font in a given  
/// environment because ``Font`` is a  
late-binding token.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct Font : Hashable,  
Sendable {  
  
    /// 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 == (lhs: Font,  
rhs: Font) -> 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 }  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Font {  
  
    /// A font with the large title text  
style.  
    public static let largeTitle: Font  
  
    /// A font with the title text style.  
    public static let title: Font  
  
    /// Create a font for second level  
hierarchical headings.  
    @available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
    public static let title2: Font
```

```
    /// Create a font for third level  
    hierarchical headings.  
    @available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
    public static let title3: Font  
  
    /// A font with the headline text  
    style.  
    public static let headline: Font  
  
    /// A font with the subheadline text  
    style.  
    public static let subheadline: Font  
  
    /// A font with the body text style.  
    public static let body: Font  
  
    /// A font with the callout text  
    style.  
    public static let callout: Font  
  
    /// A font with the footnote text  
    style.  
    public static let footnote: Font  
  
    /// A font with the caption text  
    style.  
    public static let caption: Font  
  
    /// Create a font with the alternate  
    caption text style.  
    @available(iOS 14.0, macOS 11.0, tvOS
```

```
14.0, watchOS 7.0, *)
public static let caption2: Font

    /// Gets a system font that uses the
    /// specified style, design, and weight.
    ///
    /// Use this method to create a
    /// system font that has the specified
    /// properties. The following example
    /// creates a system font with the
    /// ``TextStyle/body`` text style, a
    /// ``Design/serif`` design, and
    /// a ``Weight/bold`` weight, and
    /// applies the font to a ``Text`` view
    /// using the ``View/font(_:)`` view
    modifier:
    ///
    ///
Text("Hello").font(.system(.body, design:
    .serif, weight: .bold))
    ///
    /// The `design` and `weight`
    /// parameters are both optional. If you omit
    /// either, the system uses a default
    /// value for that parameter. The
    /// default values are typically
    /// ``Design/default`` and
    /// ``Weight/regular``,
    /// respectively, but might vary
    /// depending on the context.
    @available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public static func system(_ style:
```

```
Font.TextStyle, design: Font.Design? = nil, weight: Font.Weight? = nil) -> Font

    /// Gets a system font with the given
    text style and design.
    ///
    /// This function has been
    deprecated, use the one with nullable
    `design`
    /// and `weight` instead.
    @available(iOS, introduced: 13.0,
    deprecated: 100000.0, message: "Use
    `system(_:design:weight:)` instead.")
    @available(macOS, introduced: 10.15,
    deprecated: 100000.0, message: "Use
    `system(_:design:weight:)` instead.")
    @available(tvOS, introduced: 13.0,
    deprecated: 100000.0, message: "Use
    `system(_:design:weight:)` instead.")
    @available(watchOS, introduced: 6.0,
    deprecated: 100000.0, message: "Use
    `system(_:design:weight:)` instead.")
    @available(visionOS, introduced: 1.0,
    deprecated: 100000.0, message: "Use
    `system(_:design:weight:)` instead.")
    public static func system(_ style:
Font.TextStyle, design: Font.Design
= .default) -> Font

    /// A dynamic text style to use for
    fonts.
    public enum TextStyle : CaseIterable,
    Sendable {
```

```
    /// The font style for large
titles.
    case largeTitle

        /// The font used for first level
hierarchical headings.
    case title

        /// The font used for second
level hierarchical headings.
@available(iOS 14.0, macOS 11.0,
tvOS 14.0, watchOS 7.0, *)
    case title2

        /// The font used for third level
hierarchical headings.
@available(iOS 14.0, macOS 11.0,
tvOS 14.0, watchOS 7.0, *)
    case title3

        /// The font used for headings.
    case headline

        /// The font used for
subheadings.
    case subheadline

        /// The font used for body text.
    case body

        /// The font used for callouts.
    case callout
```

```
    /// The font used in footnotes.  
    case footnote  
  
    /// The font used for standard  
    captions.  
    case caption  
  
    /// The font used for alternate  
    captions.  
    @available(iOS 14.0, macOS 11.0,  
    tvOS 14.0, watchOS 7.0, *)  
    case caption2  
  
    /// A collection of all values of  
    this type.  
    public static let allCases:  
    [Font.TextStyle]  
  
    /// 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:  
    Font.TextStyle, b: Font.TextStyle) ->
```

## Bool

```
    /// 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)
```

```
    /// A type that can represent a
    collection of all values of this type.
    @available(iOS 13.0, tvOS 13.0,
    watchOS 6.0, macOS 10.15, *)
    public typealias AllCases =
[Font.TextStyle]

        /// 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 }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Font {

    /// Adds italics to the font.
```

```
public func italic() -> Font

    /// Adjusts the font to enable all
    small capitals.
    ///
    /// See ``Font/lowercaseSmallCaps()``  
and ``Font/uppercaseSmallCaps()`` for
    /// more details.
public func smallCaps() -> Font

    /// Adjusts the font to enable
    lowercase small capitals.
    ///
    /// This function turns lowercase
    characters into small capitals for the
    /// font. It is generally used for
    display lines set in large and small
    /// caps, such as titles. It may
    include forms related to small capitals,
    /// such as old-style figures.
public func lowercaseSmallCaps() ->
Font

    /// Adjusts the font to enable
    uppercase small capitals.
    ///
    /// This feature turns capital
    characters into small capitals. It is
    /// generally used for words which
    would otherwise be set in all caps, such
    /// as acronyms, but which are
    desired in small-cap form to avoid
    disrupting
```

```
    /// the flow of text.  
    public func uppercaseSmallCaps() ->  
Font  
  
        /// Returns a modified font that uses  
fixed-width digits, while leaving  
        /// other characters proportionally  
spaced.  
        ///  
        /// This modifier only affects  
numeric characters, and leaves all other  
        /// characters unchanged. If the base  
font doesn't support fixed-width,  
        /// or monospace digits, the font  
remains unchanged.  
        ///  
        /// The following example shows two  
text fields arranged in a ``VStack``.  
        /// Both text fields specify the 12-  
point system font, with the second  
        /// adding the `monospacedDigit()`  
modifier to the font. Because the text  
        /// includes the digit 1, normally a  
narrow character in proportional  
        /// fonts, the second text field  
becomes wider than the first.  
        ///  
        ///     @State private var userText =  
"Effect of monospacing digits: 111,111."  
        ///  
        ///     var body: some View {  
        ///         VStack {  
        ///
```

```
TextField("Proportional", text:  
$userText)  
    /// .font(.system(size:  
e: 12))  
    ///  
TextField("Monospaced", text: $userText)  
    /// .font(.system(size:  
e: 12).monospacedDigit())  
    /// }  
    /// .padding()  
    /// .navigationTitle(Text("Fo  
nt + monospacedDigit())))  
    /// }  
    ///  
    /// ! [A macOS window showing two text  
fields arranged vertically. Each  
    /// shows the text Effect of  
monospacing digits: 111,111. The even  
spacing  
    /// of the digit 1 in the second text  
field causes it to be noticeably wider  
    /// than the first.] (Environment-  
Font-monospacedDigit-1)  
    ///  
    /// - Returns: A font that uses  
fixed-width numeric characters.  
public func monospacedDigit() -> Font  
  
    /// Sets the weight of the font.  
public func weight(_ weight:  
Font.Weight) -> Font  
  
    /// Sets the width of the font.
```

```
    @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
    public func width(_ width:  
Font.Width) -> Font  
  
    /// Adds bold styling to the font.  
    public func bold() -> Font  
  
    /// Returns a fixed-width font from  
the same family as the base font.  
    ///  
    /// If there's no suitable font face  
in the same family, SwiftUI  
    /// returns a default fixed-width  
font.  
    ///  
    /// The following example adds the  
`monospaced()` modifier to the default  
    /// system font, then applies this  
font to a ``Text`` view:  
    ///  
    /// struct ContentView: View {  
    ///     let myFont = Font  
    ///         .system(size: 24)  
    ///         .monospaced()  
    ///  
    ///     var body: some View {  
    ///         Text("Hello, world!")  
    ///             .font(myFont)  
    ///             .padding()  
    ///             .navigationTitle(  
"Monospaced")  
    ///     }  
}
```

```
    /**
     */
    /**
     */
    /**
     * ! [A macOS window showing the text
Hello, world in a 24-point
     * fixed-width font.] (Environment-
Font-monospaced-1)
    /**
     * SwiftUI may provide different
fixed-width replacements for standard
     * user interface fonts (such as
``Font/title``, or a system font created
     * with ``Font/system(_:_:)``)
than for those same fonts when created
     * by name with
``Font/custom(_:_:)``.
    /**
     * The ``View/font(_:)`` modifier
applies the font to all text within
     * the view. To mix fixed-width text
with other styles in the same
     * `Text` view, use the
``Text/init(_:)`` initializer to
use an
     * appropriately-styled
    /**
<doc://com.apple.documentation/documentation/Foundation/AttributedString>
     * for the text view's content. You
can use the
    /**
<doc://com.apple.documentation/documentation/Foundation/AttributedString/3796160-
```

```
init>
    /// initializer to provide a
Markdown-formatted string containing the
    /// backtick-syntax (`...) to apply
code voice to specific ranges
    /// of the attributed string.
    ///
    /// - Returns: A fixed-width font
from the same family as the base font,
    /// if one is available, and a
default fixed-width font otherwise.
    @available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
public func monospaced() -> Font

    /// Adjusts the line spacing of a
font.
    ///
    /// You can change a font's line
spacing while maintaining other
    /// characteristics of the font by
applying this modifier.
    /// For example, you can decrease
spacing of the ``body`` font by
    /// applying the ``Leading/tight``
value to it:
    ///
    ///     let myFont =
Font.body.leading(.tight)
    ///
    /// The availability of leading
adjustments depends on the font. For some
    /// fonts, the modifier has no effect
```

and returns the original font.

```
///  
/// - Parameter leading: The line  
spacing adjustment to apply.  
///  
/// - Returns: A modified font that  
uses the specified line spacing, or  
/// the original font if it doesn't  
support line spacing adjustments.  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
public func leading(_ leading:  
Font.Leading) -> Font  
  
/// A weight to use for fonts.  
@frozen public struct Weight :  
Hashable {  
  
    public static let ultraLight:  
Font.Weight  
  
    public static let thin:  
Font.Weight  
  
    public static let light:  
Font.Weight  
  
    public static let regular:  
Font.Weight  
  
    public static let medium:  
Font.Weight
```

```
    public static let semibold:  
Font.Weight
```

```
        public static let bold:  
Font.Weight
```

```
            public static let heavy:  
Font.Weight
```

```
                public static let black:  
Font.Weight
```

```
        /// 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:
Font.Weight, b: Font.Weight) -> 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 width to use for fonts that
     *      have multiple widths.
     *      @available(iOS 16.0, macOS 13.0, tvOS
     *      16.0, watchOS 9.0, *)
     *      public struct Width : Hashable,
     *      Sendable {
     *
     *          public var value: CGFloat
     *
     *          public static let compressed:
     *          Font.Width
     *
     *          public static let condensed:
     *          Font.Width
     *
     *          public static let standard:
     *          Font.Width
     *
     *          public static let expanded:
     *          Font.Width
```

```
public init(_ value: CGFloat)

    /// 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:  
Font.Width, b: Font.Width) -> 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 line spacing adjustment that
you can apply to a font.
    /**
     /// Apply one of the `Leading` values
to a font using the
     /// ``Font/leading(_:)`` method to
increase or decrease the line spacing.
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
public enum Leading : Sendable {

    /// The font's default line
spacing.
    /**
     /// If you modify a font to use a
nonstandard line spacing like
     /// ``tight`` or ``loose``, you
can use this value to return to
     /// the font's default line
spacing.

    case standard

    /// Reduced line spacing.
    /**
     /// This value typically reduces
line spacing by 1 point for watchOS
     /// and 2 points on other
platforms.

    case tight

    /// Increased line spacing.
```

```
    /**
     * This value typically
     * increases line spacing by 1 point for
     * watchOS
     * and 2 points on other
     * platforms.
     */
    case loose

        /**
         * 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: Font.Leading, b: Font.Leading) -> Bool

        /**
         * 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)

        /// 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 }  
    }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Font {  
  
    /// Create a custom font with the  
    given `name` and `size` that scales with  
    /// the body text style.  
    public static func custom(_ name:  
String, size: CGFloat) -> Font  
  
    /// Create a custom font with the  
    given `name` and `size` that scales  
    /// relative to the given  
    `textStyle`.  
    @available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
    public static func custom(_ name:  
String, size: CGFloat, relativeTo  
textStyle: Font.TextStyle) -> Font  
  
    /// Create a custom font with the  
    given `name` and a fixed `size` that does  
    /// not scale with Dynamic Type.  
    @available(iOS 14.0, macOS 11.0, tvOS
```

```
14.0, watchOS 7.0, *)
    public static func custom(_ name:
String, fixedSize: CGFloat) -> Font

        /// Creates a custom font from a
platform font instance.
        ///
        /// Initializing ``Font`` with
platform font instance
        ///
(<doc://com.apple.documentation/documentation/CoreText/CTFont-q6r>) can bridge
SwiftUI
        /// ``Font`` with
<doc://com.apple.documentation/documentation/AppKit/NSFont> or
        ///
<doc://com.apple.documentation/documentation/UIKit/UIFont>, both of which are
        /// toll-free bridged to
        ///
<doc://com.apple.documentation/documentation/CoreText/CTFont-q6r>. For example:
        ///
        ///     // Use native Core Text API
to create desired ctFont.
        ///     let ctFont =
CTFontCreateUIFontForLanguage(.system,
12, nil)!
        ///
        ///     // Create SwiftUI Text with
the CTFont instance.
        ///     let text =
```

```
Text("Hello").font(Font(ctFont))
    public init(_ font: CTFont)
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Font {

    /// Specifies a system font to use,
    along with the style, weight, and any
    /// design parameters you want
    applied to the text.
    ///
    /// Use this function to create a
    system font by specifying the size and
    /// weight, and a type design
    together. The following styles the system
    font
    /// as 17 point,
    ``Font/Weight/semibold`` text:
    ///
    ///
Text("Hello").font(.system(size: 17,
weight: .semibold))
    ///
    /// While the following styles the
    text as 17 point ``Font/Weight/bold``,
    /// and applies a `serif`
    ``Font/Design`` to the system font:
    ///
    ///
Text("Hello").font(.system(size: 17,
weight: .bold, design: .serif))
```

```
///  
/// Both `weight` and `design` can be  
optional. When you do not provide a  
/// `weight` or `design`, the system  
can pick one based on the current  
/// context, which may not be  
``Font/Weight/regular`` or  
/// ``Font/Design/default`` in  
certain context. The following example  
styles  
    /// the text as 17 point system font  
using ``Font/Design/rounded`` design,  
    /// while its weight can depend on  
the current context:  
    ///  
    ///  
Text("Hello").font(.system(size: 17,  
design: .rounded))  
    @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
    public static func system(size:  
CGFloat, weight: Font.Weight? = nil,  
design: Font.Design? = nil) -> Font  
  
    /// Specifies a system font to use,  
along with the style, weight, and any  
    /// design parameters you want  
applied to the text.  
    ///  
    /// Use this function to create a  
system font by specifying the size and  
    /// weight, and a type design  
together. The following styles the system
```

```
font
    /// as 17 point,
``Font/Weight/semibold`` text:
    ///
    ///
Text("Hello").font(.system(size: 17,
weight: .semibold))
    ///
    /// While the following styles the
text as 17 point ``Font/Weight/bold``,
    /// and applies a `serif`
``Font/Design`` to the system font:
    ///
    ///
Text("Hello").font(.system(size: 17,
weight: .bold, design: .serif))
    ///
    /// If you want to use the default
``Font/Weight``
    /// (``Font/Weight/regular``), you
don't need to specify the `weight` in the
    /// method. The following example
styles the text as 17 point
    /// ``Font/Weight/regular``, and uses
a ``Font/Design/rounded`` system font:
    ///
    ///
Text("Hello").font(.system(size: 17,
design: .rounded))
    ///
    /// This function has been
deprecated, use the one with nullable
`weight`
```

```
    /// and `design` instead.  
    @available(iOS, introduced: 13.0,  
deprecated: 100000.0, message: "Use  
`system(size:weight:design:)` instead.")  
    @available(macOS, introduced: 10.15,  
deprecated: 100000.0, message: "Use  
`system(size:weight:design:)` instead.")  
    @available(tvOS, introduced: 13.0,  
deprecated: 100000.0, message: "Use  
`system(size:weight:design:)` instead.")  
    @available(watchOS, introduced: 6.0,  
deprecated: 100000.0, message: "Use  
`system(size:weight:design:)` instead.")  
    @available(visionOS, introduced: 1.0,  
deprecated: 100000.0, message: "Use  
`system(size:weight:design:)` instead.")  
    public static func system(size:  
CGFloat, weight: Font.Weight = .regular,  
design: Font.Design = .default) -> Font  
  
    /// A design to use for fonts.  
    public enum Design : Hashable,  
Sendable {  
  
        case `default`  
  
        @available(watchOS 7.0, *)  
        case serif  
  
        case rounded  
  
        @available(watchOS 7.0, *)  
        case monospaced
```

```
    /// 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:  
Font.Design, b: Font.Design) -> Bool
```

```
    /// 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)

    /// 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 }
}
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
extension Font.TextStyle : Equatable {  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
extension Font.TextStyle : Hashable {  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
extension Font.Weight : Sendable {  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
extension Font.Weight : BitwiseCopyable {  
}
```

```
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)
```

```
extension Font.Leading : Equatable {  
}
```

```
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)
```

```
extension Font.Leading : Hashable {  
}
```

```
/// A structure that computes views on
```

```
demand from an underlying collection of
/// identified data.
///
/// Use `ForEach` to provide views based
on a
///
<doc://com.apple.documentation/documentation/Swift/RandomAccessCollection>
/// of some data type. Either the
collection's elements must conform to
///
<doc://com.apple.documentation/documentation/Swift/Identifiable> or you
/// need to provide an `id` parameter to
the `ForEach` initializer.
///
/// The following example creates a
`NamedFont` type that conforms to
///
<doc://com.apple.documentation/documentation/Swift/Identifiable>, and an
/// array of this type called
`namedFonts`. A `ForEach` instance
iterates
/// over the array, producing new
``Text`` instances that display examples
/// of each SwiftUI ``Font`` style
provided in the array.
///
///     private struct NamedFont:
Identifiable {
///         let name: String
///         let font: Font
```

```
///         var id: String { name }
///     }
///
///     private let namedFonts:
[NamedFont] = [
    ///         NamedFont(name: "Large
Title", font: .largeTitle),
    ///         NamedFont(name: "Title",
font: .title),
    ///         NamedFont(name: "Headline",
font: .headline),
    ///         NamedFont(name: "Body", font:
.body),
    ///         NamedFont(name: "Caption",
font: .caption)
]
///
///     var body: some View {
///         ForEach(namedFonts)
{ namedFont in
    ///             Text(namedFont.name)
    ///             .font(namedFont.font)
}
///
/// ! [A vertically arranged stack of
labels showing various standard fonts,
// such as Large Title and Headline.]
(SwiftUI-ForEach-fonts.png)
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public struct ForEach<Data, ID, Content>
where Data : RandomAccessCollection, ID :
```

```
Hashable {  
  
    /// The collection of underlying  
    identified data that SwiftUI uses to  
    create  
    /// views dynamically.  
    public var data: Data  
  
    /// A function to create content on  
    demand using the underlying data.  
    public var content: (Data.Element) ->  
Content  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension ForEach {  
  
    /// Creates an instance that uniquely  
    identifies and creates views across  
    /// updates based on the sections of  
    a given view.  
    ///  
    /// - Parameters:  
    ///     - view: The view to extract the  
    sections of.  
    ///     - content: The view builder  
    that creates views from sections  
    public init<V>(sections view: V,  
@ViewBuilder content: @escaping  
(SectionConfiguration) -> Content) where  
Data ==  
ForEachSectionCollection<Content>, ID ==
```

```
SectionConfiguration.ID, Content : View,  
V : View  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension ForEach {  
  
    /// Creates an instance that uniquely  
    identifies and creates views across  
    /// updates based on the subviews of  
    a given view.  
    ///  
    /// Subviews are proxies to the  
    resolved view they represent, meaning  
    /// that modifiers applied to the  
    original view will be applied before  
    /// modifiers applied to the subview,  
    and the view is resolved  
    /// using the environment of its  
    container, *not* the environment of the  
    /// its subview proxy. Additionally,  
    because subviews must represent a  
    /// single leaf view, or container, a  
    subview may represent a view after the  
    /// application of styles. As such,  
    attempting to apply a style to it may  
    /// have no affect.  
    ///  
    /// - Parameters:  
    ///     - view: The view to extract the  
    subviews of.  
    ///     - content: The view builder
```

that creates views from subviews.

```
public init<V>(subviews view: V,  
@ViewBuilder content: @escaping (Subview)  
-> Content) where Data ==  
ForEachSubviewCollection<Content>, ID ==  
Subview.ID, Content : View, V : View  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ForEach : DynamicViewContent  
where Content : View {  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ForEach : View where Content :  
View {
```

    /// The type of view representing the  
    body of this view.

```
    ///  
    /// When you create a custom view,  
    Swift infers this type from your  
    /// implementation of the required  
    ``View/body-swift.property`` property.
```

```
    public typealias Body = Never  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ForEach where ID ==  
Data.Element.ID, Content : View,
```

```
  Data.Element : Identifiable {  
  
    /// Creates an instance that uniquely  
    identifies and creates views across  
    /// updates based on the identity of  
    the underlying data.  
    ///  
    /// It's important that the `id` of a  
    data element doesn't change unless you  
    /// replace the data element with a new  
    /// identity. If the `id` of a data  
    element changes, the content view  
    /// generated from that data element  
    loses any current state and animations.  
    ///  
    /// - Parameters:  
    ///   - data: The identified data  
    that the ``ForEach`` instance uses to  
    ///     create views dynamically.  
    ///   - content: The view builder  
    that creates views dynamically.  
    public init(_ data: Data,  
    @ViewBuilder content: @escaping  
    (Data.Element) -> Content)  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ForEach where Content : View {  
  
    /// Creates an instance that uniquely  
    identifies and creates views across
```

```
    /// updates based on the provided key
path to the underlying data's
    /// identifier.
    ///
    /// It's important that the `id` of a
data element doesn't change, unless
    /// SwiftUI considers the data
element to have been replaced with a new
data
    /// element that has a new identity.
If the `id` of a data element changes,
    /// then the content view generated
from that data element will lose any
    /// current state and animations.
    ///
    /// - Parameters:
    ///   - data: The data that the
``ForEach`` instance uses to create views
    ///   dynamically.
    ///   - id: The key path to the
provided data's identifier.
    ///   - content: The view builder
that creates views dynamically.
public init(_ data: Data, id:
KeyPath<Data.Element, ID>, @ViewBuilder
content: @escaping (Data.Element) ->
Content)
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ForEach where Content : View {
```

```
    /// Creates an instance that uniquely
    identifies and creates views across
    /// updates based on the identity of
    the underlying data.
    ///
    /// It's important that the `id` of a
    data element doesn't change unless you
    /// replace the data element with a
    new data element that has a new
    /// identity. If the `id` of a data
    element changes, the content view
    /// generated from that data element
    loses any current state and animations.
    ///
    /// - Parameters:
    ///   - data: The identified data
    that the ``ForEach`` instance uses to
    ///   create views dynamically.
    ///   - content: The view builder
    that creates views dynamically.
    public init<C>(_ data: Binding<C>,
@ViewBuilder content: @escaping
(Binding<C.Element>) -> Content) where
Data == LazyMapSequence<C.Indices,
(C.Index, ID)>, ID == C.Element.ID, C :
MutableCollection, C :
RandomAccessCollection, C.Element :
Identifiable, C.Index : Hashable

    /// Creates an instance that uniquely
    identifies and creates views across
    /// updates based on the identity of
    the underlying data.
```

```
    /**
     * It's important that the `id` of a
     * data element doesn't change unless you
     * replace the data element with a new
     * data element that has a new
     * identity. If the `id` of a data
     * element changes, the content view
     * generated from that data element
     * loses any current state and animations.
     */
    /**
     * - Parameters:
     *   - data: The identified data
     *     that the ``ForEach`` instance uses to
     *       create views dynamically.
     *   - id: The key path to the
     *     provided data's identifier.
     *   - content: The view builder
     *     that creates views dynamically.
     */
    public init<C>(_ data: Binding<C>,
                    id: KeyPath<C.Element, ID>, @ViewBuilder
                    content: @escaping (Binding<C.Element>)
                    -> Content) where Data ==
        LazyMapSequence<C.Indices, (C.Index,
                                    ID)>, C : MutableCollection, C :
        RandomAccessCollection, C.Index :
        Hashable
    }

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ForEach where Data ==
Range<Int>, ID == Int, Content : View {
```

```
    /// Creates an instance that computes
    views on demand over a given constant
    /// range.
    ///
    /// The instance only reads the
    initial value of the provided `data` and
    /// doesn't need to identify views
    across updates. To compute views on
    /// demand over a dynamic range, use
    ``ForEach/init(_:_:content:)``.
    ///
    /// - Parameters:
    ///   - data: A constant range.
    ///   - content: The view builder
    that creates views dynamically.
    public init(_ data: Range<Int>,
    @ViewBuilder content: @escaping (Int) ->
    Content)
}
```

```
/// A collection which allows a view to
be treated as a collection of its
/// sections in a for each loop.
///
/// You don't use this type directly.
Instead SwiftUI creates this type on
/// your behalf.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct
ForEachSectionCollection<Content> :
RandomAccessCollection where Content :
View {
```

```
    /// The position of the first element  
    // in a nonempty collection.  
    ///  
    /// If the collection is empty,  
    `startIndex` is equal to `endIndex`.  
    public var startIndex: Int { get }  
  
    /// The collection's "past the end"  
    position---that is, the position one  
    /// greater than the last valid  
    subscript argument.  
    ///  
    /// When you need a range that  
    includes the last element of a  
    collection, use  
    /// the half-open range operator  
    (`..<`) with `endIndex`. The `..<`  
    operator  
    /// creates a range that doesn't  
    include the upper bound, so it's always  
    /// safe to use with `endIndex`. For  
    example:  
    ///  
    ///     let numbers = [10, 20, 30,  
40, 50]  
    ///     if let index =  
numbers.firstIndex(of: 30) {  
    ///         print(numbers[index ..<  
numbers.endIndex])  
    ///     }  
    ///     // Prints "[30, 40, 50]"  
    ///
```

```
    /// If the collection is empty,  
`endIndex` is equal to `startIndex`.  
    public var endIndex: Int { get }  
  
    /// Accesses the element at the  
specified position.  
    ///  
    /// The following example accesses an  
element of an array through its  
    /// subscript to print its value:  
    ///  
    ///     var streets = ["Adams",  
"Bryant", "Channing", "Douglas",  
"Evarts"]  
    ///     print(streets[1])  
    ///     // Prints "Bryant"  
    ///  
    /// You can subscript a collection  
with any valid index other than the  
    /// collection's end index. The end  
index refers to the position one past  
    /// the last element of a collection,  
so it doesn't correspond with an  
    /// element.  
    ///  
    /// - Parameter position: The  
position of the element to access.  
`position`  
    /// must be a valid index of the  
collection that is not equal to the  
    /// `endIndex` property.  
    ///  
    /// - Complexity: O(1)
```

```
    public subscript(index: Int) ->
SectionConfiguration { get }

        /// A type representing the
sequence's elements.
        @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
        public typealias Element =
SectionConfiguration

        /// A type that represents a position
in the collection.
        /**
         * Valid indices consist of the
position of every element and a
         * "past the end" position that's
not valid for use as a subscript
         * argument.
         * @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
        public typealias Index = Int

        /// A type that represents the
indices that are valid for subscripting
the
        * collection, in ascending order.
        * @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
        public typealias Indices = Range<Int>
```

```
    /// A type that provides the
    collection's iteration interface and
    /// encapsulates its iteration state.
    ///
    /// By default, a collection conforms
    to the `Sequence` protocol by
    /// supplying `IndexingIterator` as
    its associated `Iterator`
    /// type.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias Iterator =
IndexingIterator<ForEachSectionCollection
<Content>>

    /// A collection representing a
contiguous subrange of this collection's
    /// elements. The subsequence shares
indices with the original collection.
    ///
    /// The default subsequence type for
collections that don't define their own
    /// is `Slice`.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias SubSequence =
Slice<ForEachSectionCollection<Content>>
}

/// A collection which allows a view to
be treated as a collection of its
```

```
/// subviews in a for each loop.  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
public struct  
ForEachSubviewCollection<Content> :  
RandomAccessCollection where Content :  
View {  
  
    /// The position of the first element  
    // in a nonempty collection.  
    ///  
    /// If the collection is empty,  
`startIndex` is equal to `endIndex`.  
    public var startIndex: Int { get }  
  
    /// The collection's "past the end"  
position---that is, the position one  
    /// greater than the last valid  
subscript argument.  
    ///  
    /// When you need a range that  
includes the last element of a  
collection, use  
    /// the half-open range operator  
(`..<`) with `endIndex`. The `..<`  
operator  
    /// creates a range that doesn't  
include the upper bound, so it's always  
    /// safe to use with `endIndex`. For  
example:  
    ///  
    ///     let numbers = [10, 20, 30,  
40, 50]
```

```
    ///      if let index =
numbers.firstIndex(of: 30) {
    ///          print(numbers[index ..<
numbers.endIndex])
    ///
    ///      }
    ///      // Prints "[30, 40, 50]"
    ///
    /// If the collection is empty,
`endIndex` is equal to `startIndex`.
public var endIndex: Int { get }

    /// Accesses the element at the
specified position.
    ///
    /// The following example accesses an
element of an array through its
    /// subscript to print its value:
    ///
    ///     var streets = ["Adams",
"Bryant", "Channing", "Douglas",
"Evarts"]
    ///     print(streets[1])
    ///     // Prints "Bryant"
    ///
    /// You can subscript a collection
with any valid index other than the
    /// collection's end index. The end
index refers to the position one past
    /// the last element of a collection,
so it doesn't correspond with an
    /// element.
    ///
    /// - Parameter position: The
```

```
position of the element to access.  
`position`  
    /// must be a valid index of the  
collection that is not equal to the  
    /// `endIndex` property.  
    ///  
    /// - Complexity: O(1)  
    public subscript(index: Int) ->  
Subview { get }  
  
    /// A type representing the  
sequence's elements.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias Element = Subview  
  
    /// A type that represents a position  
in the collection.  
    ///  
    /// Valid indices consist of the  
position of every element and a  
    /// "past the end" position that's  
not valid for use as a subscript  
    /// argument.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias Index = Int  
  
    /// A type that represents the  
indices that are valid for subscripting  
the
```

```
    /// collection, in ascending order.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias Indices = Range<Int>  
  
    /// A type that provides the  
collection's iteration interface and  
    /// encapsulates its iteration state.  
    ///  
    /// By default, a collection conforms  
to the `Sequence` protocol by  
    /// supplying `IndexingIterator` as  
its associated `Iterator`  
    /// type.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias Iterator =  
IndexingIterator<ForEachSubviewCollection  
<Content>>  
  
    /// A collection representing a  
contiguous subrange of this collection's  
    /// elements. The subsequence shares  
indices with the original collection.  
    ///  
    /// The default subsequence type for  
collections that don't define their own  
    /// is `Slice`.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)
```

```
    public typealias SubSequence =  
Slice<ForEachSubviewCollection<Content>>  
}  
  
/// The foreground style in the current  
context.  
///  
/// You can also use  
``ShapeStyle/foreground`` to construct  
this style.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct ForegroundColor :  
ShapeStyle {  
  
    /// Creates a foreground style  
instance.  
    @inlinable public init()  
  
    /// The type of shape style this will  
resolve to.  
    ///  
    /// When you create a custom shape  
style, Swift infers this type  
    /// from your implementation of the  
required `resolve` function.  
    @available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
    public typealias Resolved = Never  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
extension ForegroundColor :  
BitwiseCopyable {  
}  
  
/// An effect that changes the visual  
appearance of a view, largely without  
/// changing its ancestors or  
descendants.  
///  
/// The only change the effect makes to  
the view's ancestors and descendants is  
/// to change the coordinate transform to  
and from them.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
public protocol GeometryEffect :  
Animatable, ViewModifier,  
_RemoveGlobalActorIsolation where  
Self.Body == Never {  
  
    /// Returns the current value of the  
effect.  
    func effectValue(size: CGSize) ->  
ProjectionTransform  
}  
  
/// A proxy for access to the size and  
coordinate space (for anchor resolution)  
/// of the container view.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
public struct GeometryProxy {
```

```
    /// The size of the container view.  
    public var size: CGSize { get }  
  
    /// Resolves the value of an anchor  
    to the container view.  
    public subscript<T>(anchor:  
        Anchor<T>) -> T { get }  
  
    /// The safe area inset of the  
    container view.  
    public var safeAreaInsets: EdgeInsets  
    { get }  
  
    /// Returns the container view's  
    bounds rectangle, converted to a defined  
    /// coordinate space.  
    @available(iOS, introduced: 13.0,  
    deprecated: 100000.0, message: "use  
    overload that accepts a  
    CoordinateSpaceProtocol instead")  
    @available(macOS, introduced: 10.15,  
    deprecated: 100000.0, message: "use  
    overload that accepts a  
    CoordinateSpaceProtocol instead")  
    @available(tvOS, introduced: 13.0,  
    deprecated: 100000.0, message: "use  
    overload that accepts a  
    CoordinateSpaceProtocol instead")  
    @available(watchOS, introduced: 6.0,  
    deprecated: 100000.0, message: "use  
    overload that accepts a  
    CoordinateSpaceProtocol instead")  
    @available(visionOS, introduced: 1.0,
```

```
deprecated: 100000.0, message: "use
overload that accepts a
CoordinateSpaceProtocol instead")
    public func frame(in coordinateSpace:
CoordinateSpace) -> CGRect
}

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

    /// Returns the given coordinate
    space's bounds rectangle, converted to
    the
        /// local coordinate space.
    public func bounds(of
coordinateSpace: NamedCoordinateSpace) ->
CGRect?

    /// Returns the container view's
    bounds rectangle, converted to a defined
        /// coordinate space.
    public func frame(in coordinateSpace:
some CoordinateSpaceProtocol) -> CGRect
}

/// A container view that defines its
content as a function of its own size and
/// coordinate space.
///
/// This view returns a flexible
preferred size to its parent layout.
@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
@frozen public struct
GeometryReader<Content> : View where
Content : View {

    public var content: (GeometryProxy)
-> Content

    @inlinable public init(@ViewBuilder
content: @escaping (GeometryProxy) ->
Content)

        /// The type of view representing the
body of this view.
        ///
        /// When you create a custom view,
Swift infers this type from your
        /// implementation of the required
``View/body-swift.property`` property.
        @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
        public typealias Body = Never
}

/// An instance that matches a sequence
of events to a gesture, and returns a
/// stream of values for each of its
states.
///
/// Create custom gestures by declaring
types that conform to the `Gesture`
/// protocol.
@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
@MainActor @preconcurrency public
protocol Gesture<Value> {

    /// The type representing the
    gesture's value.
    associatedtype Value

    /// The type of gesture representing
    the body of `Self`.
    associatedtype Body : Gesture

    /// The content and behavior of the
    gesture.
    @MainActor @preconcurrency var body:
    Self.Body { get }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Gesture {

    /// Combines a gesture with another
    gesture to create a new gesture that
    /// recognizes both gestures at the
    same time.
    ///
    /// - Parameter other: A gesture that
    you want to combine with your gesture
    /// to create a new, combined
    gesture.
    ///
    /// - Returns: A gesture with two
```

simultaneous gestures.

```
@MainActor @inlinable @preconcurrency
public func simultaneously<Other>(with
other: Other) ->
SimultaneousGesture<Self, Other> where
Other : Gesture
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Gesture {
```

    /// Combines two gestures exclusively  
    to create a new gesture where only one

        /// gesture succeeds, giving  
        precedence to the first gesture.

        ///

        /// – Parameter other: A gesture you  
        combine with your gesture, to create a

        /// new, combined gesture.

        ///

        /// – Returns: A gesture that's the  
        result of combining two gestures where

        /// only one of them can succeed.

SwiftUI gives precedence to the first

        /// gesture.

```
@MainActor @inlinable @preconcurrency
public func exclusively<Other>(before
other: Other) -> ExclusiveGesture<Self,
Other> where Other : Gesture
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
extension Gesture {

    /// Adds an action to perform when
    the gesture ends.
    ///
    /// - Parameter action: The action to
    perform when this gesture ends. The
    /// `action` closure's parameter
    contains the final value of the gesture.
    ///
    /// - Returns: A gesture that
    triggers `action` when the gesture ends.
    nonisolated public func onEnded(_
action: @escaping (Self.Value) -> Void)
-> _EndedGesture<Self>
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Gesture where Self.Value : Equatable {

    /// Adds an action to perform when
    the gesture's value changes.
    ///
    /// - Parameter action: The action to
    perform when this gesture's value
    /// changes. The `action` closure's
    parameter contains the gesture's new
    /// value.
    ///
    /// - Returns: A gesture that
```

```
triggers `action` when this gesture's
value
    /// changes.
    @MainActor @preconcurrency public
func onChanged(_ action: @escaping
(Self.Value) -> Void) ->
_ChangedGesture<Self>
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Gesture {

    /// Returns a gesture that uses the
given closure to map over this
    /// gesture's value.
    @MainActor @preconcurrency public
func map<T>(_ body: @escaping
(Self.Value) -> T) -> _MapGesture<Self,
T>
}

/// Options that control how adding a
gesture to a view affects other gestures
/// recognized by the view and its
subviews.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct GestureMask : OptionSet {

    /// The corresponding value of the
raw type.
```

```
///  
/// A new instance initialized with  
`rawValue` will be equivalent to this  
/// instance. For example:  
///  
///     enum PaperSize: String {  
///         case A4, A5, Letter,  
Legal  
///     }  
///  
///     let selectedSize =  
PaperSize.Letter  
///     print(selectedSize.rawValue)  
///     // Prints "Letter"  
///  
///     print(selectedSize ==  
PaperSize(rawValue:  
selectedSize.rawValue)!)  
///     // Prints "true"  
public let rawValue: UInt32  
  
    /// Creates a new option set from the  
given raw value.  
///  
/// This initializer always succeeds,  
even if the value passed as `rawValue`  
/// exceeds the static properties  
declared as part of the option set. This  
/// example creates an instance of  
`ShippingOptions` with a raw value beyond  
/// the highest element, with a bit  
mask that effectively contains all the  
/// declared static members.
```

```
///  
///     let extraOptions =  
ShippingOptions(rawValue: 255)  
///  
print(extraOptions.isStrictSuperset(of: .  
all))  
///     // Prints "true"  
///  
/// - Parameter rawValue: The raw  
value of the option set to create. Each  
bit  
    /// of `rawValue` potentially  
represents an element of the option set,  
    /// though raw values may include  
bits that are not defined as distinct  
    /// values of the `OptionSet` type.  
public init(rawValue: UInt32)  
  
    /// Disable all gestures in the  
subview hierarchy, including the added  
    /// gesture.  
public static let none: GestureMask  
  
    /// Enable the added gesture but  
disable all gestures in the subview  
    /// hierarchy.  
public static let gesture:  
GestureMask  
  
    /// Enable all gestures in the  
subview hierarchy but disable the added  
    /// gesture.  
public static let subviews:
```

## GestureMask

```
    /// Enable both the added gesture as
    well as all other gestures on the view
    /// and its subviews.
    public static let all: GestureMask

        /// The type of the elements of an
        array literal.
        @available(iOS 13.0, tvOS 13.0,
        watchOS 6.0, macOS 10.15, *)
        public typealias ArrayLiteralElement
= GestureMask

        /// The element type of the option
        set.
        ///
        /// To inherit all the default
        implementations from the `OptionSet`
        protocol,
        /// the `Element` type must be
        `Self`, the default.
        @available(iOS 13.0, tvOS 13.0,
        watchOS 6.0, macOS 10.15, *)
        public typealias Element =
        GestureMask

        /// The raw type that can be used to
        represent all values of the conforming
        /// type.
        ///
        /// Every distinct value of the
        conforming type has a corresponding
```

```
unique
    /// value of the `RawValue` type, but
    there may be values of the `RawValue`
    /// type that don't have a
    corresponding value of the conforming
    type.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias RawValue = UInt32
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension GestureMask : Sendable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension GestureMask : BitwiseCopyable {
}

/// The global coordinate space at the
root of the view hierarchy.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct GlobalCoordinateSpace :
CoordinateSpaceProtocol {

    public init()

    /// The resolved coordinate space.
    public var coordinateSpace:
CoordinateSpace { get }
}
```

```
}
```

```
/// A color gradient represented as an
array of color stops, each having a
/// parametric location value.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct Gradient :  
Equatable {  
  
    /// One color stop in the gradient.
    @frozen public struct Stop :  
Equatable {  
  
        /// The color for the stop.
        public var color: Color  
  
        /// The parametric location of
the stop.
        ///  
        /// This value must be in the
range ` [0, 1] ` .
        public var location: CGFloat  
  
        /// Creates a color stop with a
color and location.
        public init(color: Color,
location: CGFloat)  
  
        /// 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:  
Gradient.Stop, b: Gradient.Stop) -> Bool  
    }  
  
    /// The array of color stops.  
    public var stops: [Gradient.Stop]  
  
    /// Creates a gradient from an array  
of color stops.  
    public init(stops: [Gradient.Stop])  
  
    /// Creates a gradient from an array  
of colors.  
    ///  
    /// The gradient synthesizes its  
location values to evenly space the  
colors  
    /// along the gradient.  
    public init(colors: [Color])  
  
    /// 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: Gradient,
b: Gradient) -> Bool
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension Gradient : Hashable {

    /// 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)

    /// 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 }
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension Gradient : ShapeStyle {

    /// The type of shape style this will
    resolve to.
```

```
///  
/// When you create a custom shape  
style, Swift infers this type  
/// from your implementation of the  
required `resolve` function.  
@available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
public typealias Resolved = Never  
  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension Gradient {  
  
    /// A method of interpolating between  
    the colors in a gradient.  
    public struct ColorSpace : Hashable,  
Sendable {  
  
        /// Interpolates gradient colors  
        in the output color space.  
        public static let device:  
Gradient.ColorSpace  
  
        /// Interpolates gradient colors  
        in a perceptual color space.  
        public static let perceptual:  
Gradient.ColorSpace  
  
        /// 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:
Gradient.ColorSpace, b:
Gradient.ColorSpace) -> 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 }

    /**
     * Returns a version of the gradient
that will use a specified
    /**
     * color space for interpolating
```

between its colors.

```
    /**
     /**
     Rectangle().fill(.linearGradient(
        /**
         colors:
        [.white, .blue]).colorSpace(.perceptual))
        /**
        /**
        - Parameters:
        /**
         - space: The color space the
        new gradient will use to
        /**
         interpolate its constituent
        colors.
        /**
        /**
        - Returns: A new gradient that
        interpolates its colors in the
        /**
         specified color space.
        /**
        public func colorSpace(_ space:
        Gradient.ColorSpace) -> AnyGradient
    }
```

```
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension Gradient.Stop : Hashable {
```

```
    /**
     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)
```

```
    /// 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 }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Gradient.Stop : Sendable {  
}  
  
/// An immediate mode drawing  
destination, and its current state.  
///  
/// Use a context to execute 2D drawing  
primitives. For example, you can draw  
/// filled shapes using the  
``fill(_:with:style:)`` method inside a  
``Canvas``  
/// view:  
///  
///     Canvas { context, size in  
///         context.fill(  
///             Path(ellipseIn:  
CGRect(origin: .zero, size: size)),  
///             with: .color(.green))  
///     }  
///     .frame(width: 300, height: 200)  
///  
/// The example above draws an ellipse  
that just fits inside a canvas that's  
/// constrained to 300 points wide and  
200 points tall:  
///  
/// ! [A screenshot of a view that shows a
```

```
green ellipse.](GraphicsContext-1)
///
/// In addition to outlining or filling
paths, you can draw images, text,
/// and SwiftUI views. You can also use
the context to perform many common
/// graphical operations, like adding
masks, applying filters and
/// transforms, and setting a blend mode.
For example you can add
/// a mask using the
``clip(to:style:options:)`` method:
///
///     let halfSize =
size.applying(CGAffineTransform(scaleX:
0.5, y: 0.5))
///     context.clip(to:
Path(CGRect(origin: .zero, size:
halfSize)))
///     context.fill(
///         Path(ellipseIn:
CGRect(origin: .zero, size: size)),
///         with: .color(.green))
///
/// The rectangular mask hides all but
one quadrant of the ellipse:
///
/// ! [A screenshot of a view that shows
the upper left quarter of a green
/// ellipse.] (GraphicsContext-2)
///
/// The order of operations matters.
Changes that you make to the state of
```

```
/// the context, like adding a mask or a
filter, apply to later
/// drawing operations. If you reverse
the fill and clip operations in
/// the example above, so that the fill
comes first, the mask doesn't
/// affect the ellipse.
///
/// Each context references a particular
layer in a tree of transparency layers,
/// and also contains a full copy of the
drawing state. You can modify the
/// state of one context without
affecting the state of any other, even if
/// they refer to the same layer. For
example you can draw the masked ellipse
/// from the previous example into a copy
of the main context, and then add a
/// rectangle into the main context:
///
///     // Create a copy of the context
to draw a clipped ellipse.
///     var maskedContext = context
///     let halfSize =
size.applying(CGAffineTransform(scaleX:
0.5, y: 0.5))
///     maskedContext.clip(to:
Path(CGRect(origin: .zero, size:
halfSize)))
///     maskedContext.fill(
///             Path(ellipseIn:
CGRect(origin: .zero, size: size)),
///             with: .color(.green))
```

```
///  
///     // Go back to the original  
context to draw the rectangle.  
///     let origin = CGPoint(x:  
size.width / 4, y: size.height / 4)  
///     context.fill(  
///         Path(CGRect(origin: origin,  
size: halfSize)),  
///         with: .color(.blue))  
///  
/// The mask doesn't clip the rectangle  
because the mask isn't part of the  
/// main context. However, both contexts  
draw into the same view because  
/// you created one context as a copy of  
the other:  
///  
/// ! [A screenshot of a view that shows  
the upper left quarter of a green  
/// ellipse, overlaid by a blue rectangle  
centered in the  
/// view.] (GraphicsContext-3)  
///  
/// The context has access to an  
``EnvironmentValues`` instance called  
/// ``environment`` that's initially  
copied from the environment of its  
/// enclosing view. SwiftUI uses  
environment values --- like the display  
/// resolution and color scheme --- to  
resolve types like ``Image`` and  
/// ``Color`` that appear in the context.  
You can also access values stored
```

```
/// in the environment for your own
purposes.
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
@frozen public struct GraphicsContext {

    /// The ways that a graphics context
    combines new content with background
    /// content.
    ///
    /// Use one of these values to set
    the
    /// ``GraphicsContext/blendMode-
    swift.property`` property of a
    /// ``GraphicsContext``. The value
    that you set affects how content
    /// that you draw replaces or
    combines with content that you
    /// previously drew into the context.
    ///
    @frozen public struct BlendMode : RawRepresentable, Equatable {

        /// The corresponding value of
        the raw type.
        ///
        /// A new instance initialized
        with `rawValue` will be equivalent to
        this
        /// instance. For example:
        ///
        ///     enum PaperSize: String {
        ///         case A4, A5, Letter,
```

```
Legal
    /**
     */
    /**
     */
    let selectedSize =
PaperSize.Letter
    /**
print(selectedSize.rawValue)
    /**
        // Prints "Letter"
    /**
        print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    /**
        // Prints "true"
public let rawValue: Int32

    /**
Creates a new instance with
the specified raw value.
    /**
        /**
If there is no value of the
type that corresponds with the specified
raw
    /**
        /**
value, this initializer
returns `nil`. For example:
    /**
    /**
enum PaperSize: String {
    /**
        case A4, A5, Letter,
Legal
    /**
}
    /**
    /**
print(PaperSize(rawValue:
"Legal"))
    /**
        // Prints
"Optional("PaperSize.Legal")"
```

```
    /**
     *      print(PaperSize(rawValue:
    "Tabloid"))
     *      // Prints "nil"
     *
     *      /**
     * - Parameter rawValue: The raw
     * value to use for the new instance.
     * @inlinable public init(rawValue:
    Int32)

            /**
             * A mode that paints source
             * image samples over the background image
             * samples.
             */
            /**
             * This is the default blend
mode.

@inlinable public static var
normal: GraphicsContext.BlendMode { get }

            /**
             * A mode that multiplies the
             * source image samples with the background
             * image samples.
             */
            /**
             * Drawing in this mode results
             * in colors that are at least as
             * dark as either of the two
             * contributing sample colors.
@inlinable public static var
multiply: GraphicsContext.BlendMode { get }

            /**
             * A mode that multiplies the
             * inverse of the source image samples with
```

```
    /// the inverse of the background
image samples.
    /**
     /// Drawing in this mode results
in colors that are at least as light
     /// as either of the two
contributing sample colors.
@inlinable public static var
screen: GraphicsContext.BlendMode { get }

    /// A mode that either multiplies
or screens the source image samples
    /// with the background image
samples, depending on the background
    /// color.
    /**
     /// Drawing in this mode overlays
the existing image samples
    /// while preserving the
highlights and shadows of the
    /// background. The background
color mixes with the source
    /// image to reflect the
lightness or darkness of the
    /// background.
@inlinable public static var
overlay: GraphicsContext.BlendMode {
get }

    /// A mode that creates composite
image samples by choosing the darker
    /// samples from either the
source image or the background.
```

```
    /**
     * When you draw in this mode,
     * source image samples that are darker
     * than the background replace
     * the background.
     * Otherwise, the background
     * image samples remain unchanged.
     @inlinable public static var
darker: GraphicsContext.BlendMode { get }

    /**
     * A mode that creates composite
     * image samples by choosing the lighter
     * samples from either the
     * source image or the background.
     */
    /**
     * When you draw in this mode,
     * source image samples that are lighter
     * than the background replace
     * the background.
     * Otherwise, the background
     * image samples remain unchanged.
     @inlinable public static var
lighten: GraphicsContext.BlendMode {
get }

    /**
     * A mode that brightens the
     * background image samples to reflect the
     * source image samples.
     */
    /**
     * Source image sample values
     * that
     * specify black do not produce
     * a change.
```

```
    @inlinable public static var  
colorDodge: GraphicsContext.BlendMode {  
get }
```

```
        /// A mode that darkens  
background image samples to reflect the  
source
```

```
        /// image samples.
```

```
        ///
```

```
        /// Source image sample values  
that specify
```

```
        /// white do not produce a  
change.
```

```
    @inlinable public static var  
colorBurn: GraphicsContext.BlendMode {  
get }
```

```
        /// A mode that either darkens or  
lightens colors, depending on the
```

```
        /// source image sample color.
```

```
        ///
```

```
        /// If the source image sample  
color is
```

```
        /// lighter than 50% gray, the  
background is lightened, similar
```

```
        /// to dodging. If the source  
image sample color is darker than
```

```
        /// 50% gray, the background is  
darkened, similar to burning.
```

```
        /// If the source image sample  
color is equal to 50% gray, the
```

```
        /// background is not changed.  
Image samples that are equal to
```

```
    /// pure black or pure white
produce darker or lighter areas,
    /// but do not result in pure
black or white. The overall
    /// effect is similar to what
you'd achieve by shining a
    /// diffuse spotlight on the
source image. Use this to add
    /// highlights to a scene.
@inlinable public static var
softLight: GraphicsContext.BlendMode {
get }
```

  

```
    /// A mode that either multiplies
or screens colors, depending on the
    /// source image sample color.
    ///
    /// If the source image sample
color
    /// is lighter than 50% gray, the
background is lightened,
    /// similar to screening. If the
source image sample color is
    /// darker than 50% gray, the
background is darkened, similar
    /// to multiplying. If the source
image sample color is equal
    /// to 50% gray, the source image
is not changed. Image samples
    /// that are equal to pure black
or pure white result in pure
    /// black or white. The overall
effect is similar to what you'd
```

```
        /// achieve by shining a harsh
spotlight on the source image.
        /// Use this to add highlights to
a scene.
    @inlinable public static var
hardLight: GraphicsContext.BlendMode {
get }

        /// A mode that subtracts the
brighter of the source image sample color
        /// or the background image
sample color from the other.
        ///
        /// Source image sample values
that are black produce no change; white
        /// inverts the background color
values.
    @inlinable public static var
difference: GraphicsContext.BlendMode {
get }

        /// A mode that produces an
effect similar to that produced by the
        /// difference blend mode, but
with lower contrast.
        ///
        /// Source image sample values
that are black don't produce a change;
        /// white inverts the background
color values.
    @inlinable public static var
exclusion: GraphicsContext.BlendMode {
get }
```

```
    /// A mode that uses the
    luminance and saturation values of the
    /// background with the hue of
    the source image.
    @inlinable public static var hue:
GraphicsContext.BlendMode { get }

    /// A mode that uses the
    luminance and hue values of the
    background with
    /// the saturation of the source
    image.
    ///
    /// Areas of the background that
    have no saturation --- namely,
    /// pure gray areas --- don't
    produce a change.
    @inlinable public static var
saturation: GraphicsContext.BlendMode {
get }

    /// A mode that uses the
    luminance values of the background with
    the hue
    /// and saturation values of the
    source image.
    ///
    /// This mode preserves the gray
    levels in the image. You can use this
    /// mode to color monochrome
    images or to tint color images.
    @inlinable public static var
```

```
color: GraphicsContext.BlendMode { get }

        /// A mode that uses the hue and
saturation of the background with the
        /// luminance of the source
image.
        ///
        /// This mode creates an effect
that is inverse to the effect created
        /// by the ``color`` mode.
    @inlinable public static var
luminosity: GraphicsContext.BlendMode {
get }

        /// A mode that clears any pixels
that the source image overwrites.
        ///
        /// With this mode, you can use
the source image like an eraser.
        ///
        /// This mode implements the
equation `R = 0` where
        /// `R` is the composite image.
    @inlinable public static var
clear: GraphicsContext.BlendMode { get }

        /// A mode that replaces
background image samples with source
image
        /// samples.
        ///
        /// Unlike the ``normal`` mode,
the source image completely replaces
```

```
    /// the background, so that even
transparent pixels in the source image
    /// replace opaque pixels in the
background, rather than letting the
    /// background show through.
    ///
    /// This mode implements the
equation  $R = S$  where
    /// *  $R$  is the composite image.
    /// *  $S$  is the source image.
@inlinable public static var
copy: GraphicsContext.BlendMode { get }

    /// A mode that you use to paint
the source image, including
    /// its transparency, onto the
opaque parts of the background.
    ///
    /// This mode implements the
equation  $R = S \cdot Da$  where
    /// *  $R$  is the composite image.
    /// *  $S$  is the source image.
    /// *  $Da$  is the source
background's alpha value.
@inlinable public static var
sourceIn: GraphicsContext.BlendMode { get
}

    /// A mode that you use to paint
the source image onto the
    /// transparent parts of the
background, while erasing the background.
    ///
```

```
    /// This mode implements the  
equation `R = S*(1 - Da)` where  
    /// * `R` is the composite image.  
    /// * `S` is the source image.  
    /// * `Da` is the source  
background's alpha value.
```

```
    @inlinable public static var  
sourceOut: GraphicsContext.BlendMode {  
get }
```

```
    /// A mode that you use to paint  
the opaque parts of the  
    /// source image onto the opaque  
parts of the background.
```

```
    ///  
    /// This mode implements the  
equation `R = S*Da + D*(1 - Sa)` where  
    /// * `R` is the composite image.  
    /// * `S` is the source image.  
    /// * `D` is the background.  
    /// * `Sa` is the source image's  
alpha value.
```

```
    /// * `Da` is the source  
background's alpha value.
```

```
    @inlinable public static var  
sourceAtop: GraphicsContext.BlendMode {  
get }
```

```
    /// A mode that you use to paint  
the source image under  
    /// the background.
```

```
    ///  
    /// This mode implements the
```

```
equation `R = S*(1 - Da) + D` where
    /// * `R` is the composite image.
    /// * `S` is the source image.
    /// * `D` is the background.
    /// * `Da` is the source
background's alpha value.
```

```
    @inlinable public static var
destinationOver: GraphicsContext.BlendMode { get }
```

```
        /// A mode that you use to erase
any of the background that
        /// isn't covered by opaque
source pixels.
```

```
        ///
        /// This mode implements the
equation `R = D*Sa` where
        /// * `R` is the composite image.
        /// * `S` is the source image.
        /// * `Da` is the source
background's alpha value.
```

```
    @inlinable public static var
destinationIn: GraphicsContext.BlendMode
{ get }
```

```
        /// A mode that you use to erase
any of the background that
        /// is covered by opaque source
pixels.
```

```
        ///
        /// This mode implements the
equation `R = D*(1 - Sa)` where
        /// * `R` is the composite image.
```

```
    /// * `D` is the background.  
    /// * `Sa` is the source image's  
alpha value.  
    @inlinable public static var  
destinationOut: GraphicsContext.BlendMode  
{ get }  
  
        /// A mode that you use to paint  
the source image under  
        /// the background, while erasing  
any of the background not matched  
        /// by opaque pixels from the  
source image.  
        ///  
        /// This mode implements the  
equation `R = S*(1 - Da) + D*Sa` where  
        /// * `R` is the composite image.  
        /// * `S` is the source image.  
        /// * `D` is the background.  
        /// * `Sa` is the source image's  
alpha value.  
        /// * `Da` is the source  
background's alpha value.  
    @inlinable public static var  
destinationAttop:  
GraphicsContext.BlendMode { get }  
  
        /// A mode that you use to clear  
pixels where both the source and  
        /// background images are opaque.  
        ///  
        /// This mode implements the  
equation `R = S*(1 - Da) + D*(1 - Sa)`
```

where

```
    /// * `R` is the composite image.  
    /// * `S` is the source image.  
    /// * `D` is the background.  
    /// * `Sa` is the source image's  
alpha value.
```

```
    /// * `Da` is the source  
background's alpha value.
```

```
    ///
```

```
    /// This XOR mode is only  
nominally related to the classical bitmap
```

```
    /// XOR operation, which SwiftUI  
doesn't support.
```

```
    @inlinable public static var xor:  
GraphicsContext.BlendMode { get }
```

```
    /// A mode that adds the inverse  
of the color components of the source
```

```
    /// and background images, and  
then inverts the result, producing
```

```
    /// a darkened composite.
```

```
    ///
```

```
    /// This mode implements the  
equation `R = MAX(0, 1 - ((1 - D) + (1 -  
S)))` where
```

```
    /// * `R` is the composite image.
```

```
    /// * `S` is the source image.
```

```
    /// * `D` is the background.
```

```
    @inlinable public static var  
plusDarker: GraphicsContext.BlendMode {  
get }
```

```
    /// A mode that adds the
```

```
components of the source and background  
images,
```

```
    /// resulting in a lightened  
composite.
```

```
///
```

```
/// This mode implements the  
equation `R = MIN(1, S + D)` where
```

```
/// * `R` is the composite image.
```

```
/// * `S` is the source image.
```

```
/// * `D` is the background.
```

```
@inlinable public static var  
plusLighter: GraphicsContext.BlendMode {  
get }
```

```
    /// The raw type that can be used  
to represent all values of the conforming
```

```
    /// type.
```

```
///
```

```
    /// Every distinct value of the  
conforming type has a corresponding  
unique
```

```
    /// value of the `RawValue` type,  
but there may be values of the `RawValue`
```

```
    /// type that don't have a  
corresponding value of the conforming  
type.
```

```
@available(iOS 15.0, tvOS 15.0,  
watchOS 8.0, macOS 12.0, *)  
public typealias RawValue = Int32  
}
```

```
    /// The opacity of drawing operations  
in the context.
```

```
///  
/// Set this value to affect the  
opacity of content that you subsequently  
/// draw into the context. Changing  
this value has no impact on the  
/// content you previously drew into  
the context.
```

```
public var opacity: Double
```

```
/// The blend mode used by drawing  
operations in the context.
```

```
///  
/// Set this value to affect how any  
content that you subsequently draw  
/// into the context blends with  
content that's already in the context.  
/// Use one of the  
``GraphicsContext/BlendMode-  
swift.struct`` values.
```

```
public var blendMode:  
GraphicsContext.BlendMode
```

```
/// The environment associated with  
the graphics context.
```

```
///  
/// SwiftUI initially sets this to  
the environment of the context's  
/// enclosing view. The context uses  
values like display  
/// resolution and the color scheme  
from the environment to resolve types  
/// like ``Image`` and ``Color``. You  
can also access values stored in the
```

```
    /// environment for your own
purposes.
    public var environment:
EnvironmentValues { get }

    /// The current transform matrix,
defining user space coordinates.
    /**
     /// Modify this matrix to transform
content that you subsequently
     /// draw into the context. Changes
that you make don't affect
     /// existing content.
    public var transform:
CGAffineTransform

    /// Scales subsequent drawing
operations by an amount in each
dimension.
    /**
     /// Calling this method is equivalent
to updating the context's
     /// ``transform`` directly using the
given scale factors:
    /**
     ///     transform =
transform.scaledBy(x: x, y: y)
    /**
     /// - Parameters:
     ///   - x: The amount to scale in the
horizontal direction.
     ///   - y: The amount to scale in the
vertical direction.
```

```
    public mutating func scaleBy(x:  
CGFloat, y: CGFloat)  
  
        /// Moves subsequent drawing  
operations by an amount in each  
dimension.  
        ///  
        /// Calling this method is equivalent  
to updating the context's  
        /// ``transform`` directly using the  
given translation amount:  
        ///  
        ///     transform =  
transform.translatedBy(x: x, y: y)  
        ///  
        /// - Parameters:  
        ///     - x: The amount to move in the  
horizontal direction.  
        ///     - y: The amount to move in the  
vertical direction.  
    public mutating func translateBy(x:  
CGFloat, y: CGFloat)  
  
        /// Rotates subsequent drawing  
operations by an angle.  
        ///  
        /// Calling this method is equivalent  
to updating the context's  
        /// ``transform`` directly using the  
`angle` parameter:  
        ///  
        ///     transform =  
transform.rotated(by: angle.radians)
```

```
///  
/// - Parameters:  
///   - angle: The amount to rotate.  
public mutating func rotate(by angle:  
Angle)
```

```
/// Appends the given transform to  
the context's existing transform.  
///  
/// Calling this method is equivalent  
to updating the context's  
/// ``transform`` directly using the  
`matrix` parameter:  
///  
///     transform =  
matrix.concatenate(transform)  
///  
/// - Parameter matrix: A transform  
to append to the existing transform.  
public mutating func concatenate(_  
matrix: CGAffineTransform)  
  
/// Options that affect the use of  
clip shapes.  
///  
/// Use these options to affect how  
SwiftUI interprets a clip shape  
/// when you call  
``clip(to:style:options:)`` to add a path  
to the array of  
/// clip shapes, or when you call  
``clipToLayer(opacity:options:content:)``  
/// to add a clipping layer.
```

```
    @frozen public struct ClipOptions : OptionSet {  
  
        /// The corresponding value of  
        the raw type.  
        ///  
        /// A new instance initialized  
        with `rawValue` will be equivalent to  
        this  
        /// instance. For example:  
        ///  
        ///     enum PaperSize: String {  
        ///         case A4, A5, Letter,  
Legal  
        ///     }  
        ///  
        ///     let selectedSize =  
PaperSize.Letter  
        ///  
print(selectedSize.rawValue)  
        ///     // Prints "Letter"  
        ///  
        ///     print(selectedSize ==  
PaperSize(rawValue:  
selectedSize.rawValue)!)  
        ///     // Prints "true"  
    public let rawValue: UInt32  
  
        /// Creates a new option set from  
        the given raw value.  
        ///  
        /// This initializer always  
succeeds, even if the value passed as
```

```
`rawValue`  
    /// exceeds the static properties  
declared as part of the option set. This  
    /// example creates an instance  
of `ShippingOptions` with a raw value  
beyond  
    /// the highest element, with a  
bit mask that effectively contains all  
the  
    /// declared static members.  
    ///  
    ///     let extraOptions =  
ShippingOptions(rawValue: 255)  
    ///  
print(extraOptions.isStrictSuperset(of: .  
all))  
    ///     // Prints "true"  
    ///  
    /// - Parameter rawValue: The raw  
value of the option set to create. Each  
bit  
    ///     of `rawValue` potentially  
represents an element of the option set,  
    ///     though raw values may  
include bits that are not defined as  
distinct  
    ///     values of the `OptionSet`  
type.  
    @inlinable public init(rawValue:  
UInt32)  
  
    /// An option to invert the shape  
or layer alpha as the clip mask.
```

```
    /**
     * When you use this option,
SwiftUI uses `1 - alpha` instead of
     * `alpha` for the given clip
shape.

    @inlinable public static var
inverse: GraphicsContext.ClipOptions {
get }

        /**
         * The type of the elements of
an array literal.
        @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
        public typealias
ArrayLiteralElement =
GraphicsContext.ClipOptions

        /**
         * The element type of the
option set.
        /**
         * To inherit all the default
implementations from the `OptionSet`
protocol,
        /**
         * the `Element` type must be
`Self`, the default.
        @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
        public typealias Element =
GraphicsContext.ClipOptions

        /**
         * The raw type that can be used
to represent all values of the conforming
         * type.
```

```
    /**
     * Every distinct value of the
     * conforming type has a corresponding
     * unique
     *   /**
     * value of the `RawValue` type,
     * but there may be values of the `RawValue`
     *   /**
     * type that don't have a
     * corresponding value of the conforming
     * type.
     * @available(iOS 15.0, tvOS 15.0,
     * watchOS 8.0, macOS 12.0, *)
     * public typealias RawValue =
     * UInt32
     */

    /**
     * The bounding rectangle of the
     * intersection of all current clip
     * shapes in the current user space.
     * public var clipBoundingRect: CGRect {
     * get }

    /**
     * Adds a path to the context's
     * array of clip shapes.
     */
    /**
     * Call this method to add a shape
     * to the array of clip shapes that
     * the context uses to define a
     * clipping mask. Shapes that you add
     * affect only subsequent drawing
     * operations.
     */
    /**
     * - Parameters:
     *   - path: A ``Path`` that defines
```

the shape of the clipping mask.

    /// - style: A ``FillStyle`` that defines how to rasterize the shape.

    /// - options: Clip options that tell SwiftUI how to interpret the `path`

        /// as a clip shape. For example, you can invert the clip

        /// shape by setting the ``ClipOptions/inverse`` option.

```
public mutating func clip(to path: Path, style: FillStyle = FillStyle(), options: GraphicsContext.ClipOptions = ClipOptions())
```

    /// Adds a clip shape that you define in a new layer to the context's array

        /// of clip shapes.

    ///

        /// Call this method to add a shape to the array of clip shapes that

        /// the context uses to define a clipping mask. Shapes that you add

        /// affect only subsequent drawing operations.

    ///

    /// - Parameters:

        /// - opacity: A value that SwiftUI uses to multiply the alpha channel of

        /// the rasterized layer that you define in the `content` closure. The

        /// alpha values that result define the clip shape.

    /// - options: A set of options

that tell SwiftUI how to interpret the  
    ///     clip shape. For example, you  
can invert the clip  
    ///     shape by setting the  
``ClipOptions/inverse`` option.  
    /// - content: A closure that  
receives as input a new  
``GraphicsContext``,  
    ///     which represents a new  
transparency layer. The alpha channel of  
    ///     content that you draw into  
this context, multiplied by the `opacity`  
    ///     parameter, defines the clip  
shape.

```
public mutating func  
clipToLayer(opacity: Double = 1, options:  
GraphicsContext.ClipOptions =  
ClipOptions(), content: (inout  
GraphicsContext) throws -> Void) rethrows
```

    /// A type that applies image  
processing operations to rendered  
content.  
    ///  
    /// Create and configure a filter  
that produces an image processing effect,  
    /// like adding a drop shadow or a  
blur effect, by calling one of the  
    /// factory methods defined by the  
`Filter` structure. Call the  
    /// ``addFilter(\_:options:)`` method  
to add the filter to a  
    /// ``GraphicsContext``. The filter

```
only affects content that you draw
    /// into the context after adding the
filter.
public struct Filter : Sendable {

    /// Returns a filter that transforms the rasterized form
    /// of subsequent graphics primitives.
    ///
    /// - Parameters:
    ///   - matrix: A projection transform to apply to the rasterized
    ///             form of graphics primitives.
    /// - Returns: A filter that applies a transform.
    public static func
projectionTransform(_ matrix:
ProjectionTransform) ->
GraphicsContext.Filter

    /// Returns a filter that adds a shadow.
    ///
    /// SwiftUI produces the shadow by blurring the alpha channel of the
    /// object receiving the shadow, multiplying the result by a color,
    /// optionally translating the shadow by an amount,
    /// and then blending the resulting shadow into a new layer below
```

the

```
    /// source primitive. You can
    // customize some of these steps by adding
    /// one or more shadow options.
    ///
    /// - Parameters:
    ///   - color: A ``Color`` that
    tints the shadow.
    ///   - radius: A measure of how
    far the shadow extends from the edges
    ///       of the content receiving
    the shadow.
    ///   - x: An amount to translate
    the shadow horizontally.
    ///   - y: An amount to translate
    the shadow vertically.
    ///   - blendMode: The
    ``GraphicsContext/BlendMode-
    swift.struct`` to use
    ///       when blending the shadow
    into the background layer.
    ///   - options: A set of options
    that you can use to customize the
    ///       process of adding the
    shadow. Use one or more of the options
    ///       in
    ``GraphicsContext/ShadowOptions``.
    /// - Returns: A filter that adds
    a shadow style.
    public static func shadow(color:
    Color = Color(.sRGBLinear, white: 0,
    opacity: 0.33), radius: CGFloat, x:
    CGFloat = 0, y: CGFloat = 0, blendMode:
```

```
GraphicsContext.BlendMode = .normal,  
options: GraphicsContext.ShadowOptions =  
ShadowOptions()) ->  
GraphicsContext.Filter
```

```
    /// Returns a filter that  
    multiplies each color component by  
        /// the matching component of a  
    given color.  
    ///  
    /// – Parameters:  
    /// – color: The color that the  
filter uses for the multiplication  
    /// operation.  
    /// – Returns: A filter that  
multiplies color components.  
    public static func  
colorMultiply(_ color: Color) ->  
GraphicsContext.Filter
```

```
    /// Returns a filter that  
    multiplies by a given color matrix.  
    ///  
    /// This filter is equivalent to  
the `feColorMatrix` filter primitive  
    /// defined by the Scalable  
Vector Graphics (SVG) specification.  
    ///  
    /// The filter creates the output  
color `[R', G', B', A']` at each pixel  
    /// from an input color `[R, G,  
B, A]` by multiplying the input color by  
    /// the square matrix formed by
```

the first four columns of the  
    /// ``ColorMatrix``, then adding  
the fifth column to the result:

```
    ///
    ///      R' = r1 × R + r2 × G + r3
× B + r4 × A + r5
    ///      G' = g1 × R + g2 × G + g3
× B + g4 × A + g5
    ///      B' = b1 × R + b2 × G + b3
× B + b4 × A + b5
    ///      A' = a1 × R + a2 × G + a3
× B + a4 × A + a5
    ///
```

/// – Parameters:  
 /// – matrix: A ``ColorMatrix``  
instance used by the filter.

/// – Returns: A filter that  
transforms color using the given matrix.

```
public static func colorMatrix(_
matrix: ColorMatrix) ->
GraphicsContext.Filter
```

/// Returns a filter that applies  
a hue rotation adjustment.

```
    ///
    /// This filter is equivalent to
the `hue-rotate` filter primitive
    /// defined by the Scalable
Vector Graphics (SVG) specification.
```

```
    ///
    /// – Parameters:
    /// – angle: The amount by
which to rotate the hue value of each
```

```
    /// pixel.  
    /// - Returns: A filter that  
    applies a hue rotation adjustment.  
    public static func hueRotation(_  
angle: Angle) -> GraphicsContext.Filter  
  
        /// Returns a filter that applies  
        a saturation adjustment.  
        ///  
        /// This filter is equivalent to  
        the `saturate` filter primitive  
        /// defined by the Scalable  
        Vector Graphics (SVG) specification.  
        ///  
        /// - Parameters:  
        /// - amount: The amount of the  
        saturation adjustment. A value  
        /// of zero to completely  
        desaturates each pixel, while a value of  
        /// one makes no change. You  
        can use values greater than one.  
        /// - Returns: A filter that  
        applies a saturation adjustment.  
        public static func saturation(_  
amount: Double) -> GraphicsContext.Filter  
  
        /// Returns a filter that applies  
        a brightness adjustment.  
        ///  
        /// This filter is different than  
        `brightness` filter primitive  
        /// defined by the Scalable  
        Vector Graphics (SVG) specification.
```

```
    /// You can obtain an effect like
that filter using a ``grayscale(_)``
    /// color multiply. However, this
filter does match the
    ///
<doc://com.apple.documentation/documentation/CoreImage/CIColorControls>
    /// filter's brightness
adjustment.
    ///
    /// - Parameters:
    ///   - amount: An amount to add
to the pixel's color components.
    /// - Returns: A filter that
applies a brightness adjustment.
public static func brightness(_
amount: Double) -> GraphicsContext.Filter

    /// Returns a filter that applies
a contrast adjustment.
    ///
    /// This filter is equivalent to
the `contrast` filter primitive
    /// defined by the Scalable
Vector Graphics (SVG) specification.
    ///
    /// - Parameters:
    ///   - amount: An amount to
adjust the contrast. A value of
        /// zero leaves the result
completely gray. A value of one leaves
        /// the result unchanged. You
can use values greater than one.
```

```
    /// - Returns: A filter that
  applies a contrast adjustment.
  public static func contrast(_
amount: Double) -> GraphicsContext.Filter

    /// Returns a filter that inverts
the color of their results.
    ///
    /// This filter is equivalent to
the `invert` filter primitive
    /// defined by the Scalable
Vector Graphics (SVG) specification.
    ///
    /// - Parameters:
    ///   - amount: The inversion
amount. A value of one results in total
    ///   inversion, while a value
of zero leaves the result unchanged.
    ///   Other values apply a
linear multiplier effect.
    /// - Returns: A filter that
applies a color inversion.
  public static func colorInvert(_
amount: Double = 1) ->
GraphicsContext.Filter

    /// Returns a filter that applies
a grayscale adjustment.
    ///
    /// This filter is equivalent to
the `grayscale` filter primitive
    /// defined by the Scalable
Vector Graphics (SVG) specification.
```

```
///  
/// - Parameters:  
///   - amount: An amount that  
controls the effect. A value of one  
     ///   makes the image  
completely gray. A value of zero leaves  
the  
     ///   result unchanged. Other  
values apply a linear multiplier effect.  
     /// - Returns: A filter that  
applies a grayscale adjustment.  
public static func grayscale(  
amount: Double) -> GraphicsContext.Filter  
  
     /// Returns a filter that sets  
the opacity of each pixel based on its  
     /// luminance.  
     ///  
     /// The filter computes the  
luminance of each pixel  
     /// and uses it to define the  
opacity of the result, combined  
     /// with black (zero) color  
components.  
     ///  
     /// - Returns: A filter that  
applies a luminance to alpha  
transformation.  
public static var  
luminanceToAlpha: GraphicsContext.Filter  
{ get }  
  
     /// Returns a filter that applies
```

a Gaussian blur.

///

/// - Parameters:

/// - radius: The standard deviation of the Gaussian blur.

/// - options: A set of options controlling the application of the effect.

/// - Returns: A filter that applies Gaussian blur.

```
public static func blur(radius:  
CGFloat, options:  
GraphicsContext.BlurOptions =  
BlurOptions()) -> GraphicsContext.Filter
```

/// Returns a filter that replaces each pixel with alpha components within a range by a constant color, or transparency otherwise.

///

/// - Parameters:

/// - min: The minimum alpha threshold. Pixels whose alpha

/// component is less than this value will render as

/// transparent. Results are undefined unless `min < max`.

/// - max: The maximum alpha threshold. Pixels whose alpha

/// component is greater than this value will render

/// as transparent. Results are undefined unless `min < max`.

```
    /// - color: The color that is
    output for pixels with an alpha
    /// component between the two
    threshold values.
    /// - Returns: A filter that
    applies a threshold to alpha values.
    public static func
alphaThreshold(min: Double, max: Double =
1, color: Color = Color.black) ->
GraphicsContext.Filter

    /// Returns a filter that applies
`shader` to the color of each
    /// source pixel.
    ///
    /// For a shader function to act
as a color filter it must have
    /// a function signature
matching:
    ///
    ///     [[ stitchable ]] half4
name(float2 position, half4 color,
args...)
    ///
    /// where `position` is the user-
space coordinates of the pixel
    /// applied to the shader and
`color` its source color, as a
    /// pre-multiplied color in the
destination color space. `args...`
    /// should be compatible with the
uniform arguments bound to
    /// `shader`. The function should
```

```
return the modified color value.  
    ///  
    /// - Parameters:  
    ///   - shader: The shader to  
apply to `self` as a color filter.  
    ///  
    /// - Returns: A filter that  
applies the shader as a color  
    ///   filter.  
    @available(iOS 17.0, macOS 14.0,  
tvOS 17.0, *)  
    @available(watchOS, unavailable)  
    public static func colorShader(_  
shader: Shader) -> GraphicsContext.Filter  
  
        /// Returns a filter that applies  
`shader` as a geometric  
        /// distortion effect on the  
location of each pixel.  
        ///  
        /// For a shader function to act  
as a distortion effect it must  
        /// have a function signature  
matching:  
        ///  
        ///   [[ stitchable ]] float2  
name(float2 position, args...)  
        ///  
        /// where `position` is the user-  
space coordinates of the  
        /// destination pixel applied to  
the shader. `args...` should be  
        /// compatible with the uniform
```

```
arguments bound to `shader`. The
    /// function should return the
user-space coordinates of the
    /// corresponding source pixel.
    ///
    /// - Parameters:
    ///   - shader: The shader to
apply as a distortion effect.
    ///   - maxSampleOffset: The
maximum distance in each axis
    ///       between the returned
source pixel position and the
    ///       destination pixel
position, for all source pixels.
    ///
    /// - Returns: A new filter that
applies the shader as a
    ///       distortion effect.
@available(iOS 17.0, macOS 14.0,
tvOS 17.0, *)
@available(watchOS, unavailable)
public static func
distortionShader(_ shader: Shader,
maxSampleOffset: CGSize) ->
GraphicsContext.Filter

    /// Returns a filter that applies
`shader` to the contents of
    /// the source layer.
    ///
    /// For a shader function to act
as a layer effect it must
    /// have a function signature
```

matching:

```
    /**
     * [[ stitchable ]] half4
name(float2 position,
      SwiftUI::Layer layer,
args...)
    /**
     * where `position` is the user-
space coordinates of the
     * destination pixel applied to
the shader, and `layer` is a
     * rasterized subregion of the
source layer. `args...` should
     * be compatible with the
uniform arguments bound to `shader`.
    /**
     * The `SwiftUI::Layer` type is
defined in the
     * `<SwiftUI/SwiftUI.h>` header
file. It exports a single
     * `sample()` function that
returns a linearly-filtered pixel
     * value from a position in the
source content, as a
     * premultiplied RGBA pixel
value:
    /**
     * namespace SwiftUI {
     * struct Layer {
     *     half4 sample(float2
position) const;
     *         };
     *     };

```

```
    /**
     * The function should return
     * the color mapping to the
     * destination pixel, typically
     * by sampling one or more pixels
     * from `layer` at location(s)
     * derived from `position` and
     * them applying some kind of
     * transformation to produce a new
     * color.
     */
     * - Parameters:
     *   - shader: The shader to
     * apply as a layer effect.
     *   - maxSampleOffset: If the
     * shader function samples from
     * the layer at locations
     * not equal to the destination
     * position, this value must
     * specify the maximum sampling
     * distance in each axis,
     * for all source pixels.
     */
     * - Returns: A filter applies
     * the shader as a layer effect.
     */
     @available(iOS 17.0, macOS 14.0,
tvOS 17.0, *)
     @available(watchOS, unavailable)
     public static func layerShader(_
shader: Shader, maxSampleOffset: CGSize)
-> GraphicsContext.Filter
}
```

```
    /// Options that configure the
    graphics context filter that creates
    shadows.

    /**
     * You can use a set of these
     options when you call
     */
``Filter/shadow(color:radius:x:y:blendMod
e:options:)`` to create a
     * ``Filter`` that adds a drop
     shadow to an object that you draw into a
     * ``GraphicsContext``.

@frozen public struct ShadowOptions : OptionSet {

    /**
     * The corresponding value of
     the raw type.
    */
    /**
     * A new instance initialized
     with `rawValue` will be equivalent to
     this
    */
    /**
     * instance. For example:
    */
    /**
     * enum PaperSize: String {
     *     case A4, A5, Letter,
     Legal
     */
    /**
     */
    /**
     * let selectedSize =
     PaperSize.Letter
     */
print(selectedSize.rawValue)
    /**
     * // Prints "Letter"
    */
}
```

```
    /**
     *      print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
     *      // Prints "true"
public let rawValue: UInt32

        /// Creates a new option set from
the given raw value.
    /**
     * This initializer always
succeeds, even if the value passed as
`rawValue`
        /// exceeds the static properties
declared as part of the option set. This
        /// example creates an instance
of `ShippingOptions` with a raw value
beyond
        /// the highest element, with a
bit mask that effectively contains all
the
        /// declared static members.
    /**
     *      let extraOptions =
ShippingOptions(rawValue: 255)
    /**
print(extraOptions.isStrictSuperset(of: .
all))
        // Prints "true"
    /**
     * - Parameter rawValue: The raw
value of the option set to create. Each
bit
```

```
    /// of `rawValue` potentially
represents an element of the option set,
    /// though raw values may
include bits that are not defined as
distinct
    /// values of the `OptionSet`
type.
@inlinable public init(rawValue:
UInt32)

    /// An option that causes the
filter to draw the shadow above the
    /// object, rather than below it.
@inlinable public static var
shadowAbove:
GraphicsContext.ShadowOptions { get }

    /// An option that causes the
filter to draw only the shadow, and
    /// omit the source object.
@inlinable public static var
shadowOnly: GraphicsContext.ShadowOptions
{ get }

    /// An option that causes the
filter to invert the alpha of the shadow.
    ///
    /// You can create an "inner
shadow" effect by combining this option
    /// with ``shadowAbove`` and
using the
    /// ``GraphicsContext/BlendMode-
swift.struct/sourceAtop`` blend mode.
```

```
    @inlinable public static var  
invertsAlpha:  
GraphicsContext.ShadowOptions { get }  
  
        /// An option that causes the  
filter to composite the object and its  
        /// shadow separately in the  
current layer.  
    @inlinable public static var  
disablesGroup:  
GraphicsContext.ShadowOptions { get }  
  
        /// The type of the elements of  
an array literal.  
    @available(iOS 15.0, tvOS 15.0,  
watchOS 8.0, macOS 12.0, *)  
    public typealias  
ArrayLiteralElement =  
GraphicsContext.ShadowOptions  
  
        /// The element type of the  
option set.  
    ///  
    /// To inherit all the default  
implementations from the `OptionSet`  
protocol,  
        /// the `Element` type must be  
`Self`, the default.  
    @available(iOS 15.0, tvOS 15.0,  
watchOS 8.0, macOS 12.0, *)  
    public typealias Element =  
GraphicsContext.ShadowOptions
```

```
    /// The raw type that can be used
    to represent all values of the conforming
    /// type.
    ///
    /// Every distinct value of the
    conforming type has a corresponding
    unique
        /// value of the `RawValue` type,
    but there may be values of the `RawValue`
        /// type that don't have a
    corresponding value of the conforming
    type.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias RawValue =
UInt32
}

    /// Options that configure the
graphics context filter that creates
blur.
    ///
    /// You can use a set of these
options when you call
    /// ``Filter/blur(radius:options:)``
to create a ``Filter`` that adds
    /// blur to an object that you draw
into a ``GraphicsContext``.
    @frozen public struct BlurOptions : OptionSet {

        /// The corresponding value of
the raw type.
```

```
    /**
     * A new instance initialized
     * with `rawValue` will be equivalent to
     * this
     *   instance. For example:
     */
    enum PaperSize: String {
        case A4, A5, Letter,
Legal
        }
        /**
         * let selectedSize =
PaperSize.Letter
        */
print(selectedSize.rawValue)
        // Prints "Letter"
        /**
         * print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
        // Prints "true"
public let rawValue: UInt32

        /**
         * Creates a new option set from
         * the given raw value.
         */
        /**
         * This initializer always
         * succeeds, even if the value passed as
         * `rawValue`
         * exceeds the static properties
         * declared as part of the option set. This
         * example creates an instance
         * of `ShippingOptions` with a raw value
```

```
beyond
```

```
    /// the highest element, with a  
bit mask that effectively contains all  
the  
    /// declared static members.  
    ///  
    ///     let extraOptions =  
ShippingOptions(rawValue: 255)  
    ///  
print(extraOptions.isStrictSuperset(of: .  
all))  
    ///     // Prints "true"  
    ///  
    /// - Parameter rawValue: The raw  
value of the option set to create. Each  
bit  
    ///     of `rawValue` potentially  
represents an element of the option set,  
    ///     though raw values may  
include bits that are not defined as  
distinct  
    ///     values of the `OptionSet`  
type.  
    @inlinable public init(rawValue:  
UInt32)  
  
    /// An option that causes the  
filter to ensure the result is completely  
    /// opaque.  
    ///  
    /// The filter ensure opacity by  
dividing each pixel by its alpha  
    /// value. The result may be
```

```
undefined if the input to the filter
    /// isn't also completely opaque.
    @inlinable public static var
opaque: GraphicsContext.BlurOptions { get
}

        /// An option that causes the
filter to dither the result, to reduce
        /// banding.
    @inlinable public static var
dithersResult:
GraphicsContext.BlurOptions { get }

        /// The type of the elements of
an array literal.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias
ArrayLiteralElement =
GraphicsContext.BlurOptions

        /// The element type of the
option set.
    /**
     /// To inherit all the default
implementations from the `OptionSet`
protocol,
        /// the `Element` type must be
`Self`, the default.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias Element =
GraphicsContext.BlurOptions
```

```
    /// The raw type that can be used  
    to represent all values of the conforming  
    /// type.  
    ///  
    /// Every distinct value of the  
    conforming type has a corresponding  
    unique  
        /// value of the `RawValue` type,  
    but there may be values of the `RawValue`  
        /// type that don't have a  
    corresponding value of the conforming  
    type.  
    @available(iOS 15.0, tvOS 15.0,  
watchOS 8.0, macOS 12.0, *)  
    public typealias RawValue =  
UInt32  
}
```

```
    /// Options that configure a filter  
    that you add to a graphics context.  
    ///  
    /// You can use filter options to  
    configure a ``Filter`` that you apply  
    /// to a ``GraphicsContext`` with the  
    ``addFilter(_:options:)`` method.  
    @frozen public struct FilterOptions :  
OptionSet {
```

```
        /// The corresponding value of  
    the raw type.  
        ///  
        /// A new instance initialized
```

with `rawValue` will be equivalent to this

```
    /// instance. For example:  
    ///  
    ///     enum PaperSize: String {  
    ///         case A4, A5, Letter,  
Legal  
    ///     }  
    ///  
    ///     let selectedSize =  
PaperSize.Letter  
    ///  
print(selectedSize.rawValue)  
    ///     // Prints "Letter"  
    ///  
    ///     print(selectedSize ==  
PaperSize(rawValue:  
selectedSize.rawValue)!)  
    ///     // Prints "true"  
public let rawValue: UInt32  
  
    /// Creates a new option set from  
the given raw value.  
    ///  
    /// This initializer always  
succeeds, even if the value passed as  
'rawValue'  
    /// exceeds the static properties  
declared as part of the option set. This  
    /// example creates an instance  
of `ShippingOptions` with a raw value  
beyond  
    /// the highest element, with a
```

```
bit mask that effectively contains all
the
    /// declared static members.
    /**
     *      let extraOptions =
ShippingOptions(rawValue: 255)
    /**
print(extraOptions.isStrictSuperset(of: .
all))
    /**
         // Prints "true"
    /**
         /**
          * - Parameter rawValue: The raw
value of the option set to create. Each
bit
        /**
            of `rawValue` potentially
represents an element of the option set,
        /**
            though raw values may
include bits that are not defined as
distinct
        /**
            values of the `OptionSet`
type.
    @inlinable public init(rawValue:
UInt32)

        /**
            An option that causes the
filter to perform calculations in a
        /**
            linear color space.
    @inlinable public static var
linearColor:
GraphicsContext.FilterOptions { get }

        /**
            The type of the elements of
an array literal.
```

```
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
public typealias
ArrayLiteralElement =
GraphicsContext.FilterOptions

        /// The element type of the
option set.
        ///
        /// To inherit all the default
implementations from the `OptionSet`-
protocol,
        /// the `Element` type must be
`Self`, the default.
        @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
public typealias Element =
GraphicsContext.FilterOptions

        /// The raw type that can be used
to represent all values of the conforming
        /// type.
        ///
        /// Every distinct value of the
conforming type has a corresponding
unique
        /// value of the `RawValue` type,
but there may be values of the `RawValue`-
        /// type that don't have a
corresponding value of the conforming
type.
        @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
```

```
    public typealias RawValue =  
UInt32  
}  
  
    /// Adds a filter that applies to  
    subsequent drawing operations.  
    ///  
    /// To draw with filtering, SwiftUI:  
    ///  
    /// - Rasterizes the drawing  
    operation to an implicit transparency  
    layer  
    /// without blending, adjusting  
    opacity, or applying any clipping.  
    /// - Applies the filter to the layer  
    containing the rasterized image.  
    /// - Composites the layer onto the  
    background, using the context's  
    /// current blend mode, opacity  
    setting, and clip shapes.  
    ///  
    /// When SwiftUI draws with a filter,  
    the blend mode might apply to regions  
    /// outside the drawing operation's  
    intrinsic shape, but inside its clip  
    /// shape. That might result in  
    unexpected behavior for certain blend  
    /// modes like  
``GraphicsContext/BlendMode-  
swift.struct/copy``, where  
    /// the drawing operation completely  
    overwrites the background even if  
    /// the source alpha is zero.
```

```
///  
/// - Parameters:  
///   - filter: A graphics context  
filter that you create by calling one  
///     of the ``Filter`` factory  
methods.  
///   - options: A set of options  
from ``FilterOptions`` that you can use  
to  
    ///     configure filter operations.  
    public mutating func addFilter(_  
filter: GraphicsContext.Filter, options:  
GraphicsContext.FilterOptions =  
FilterOptions())  
  
    /// A color or pattern that you can  
use to outline or fill a path.  
    ///  
    /// Use a shading instance to  
describe the color or pattern of a path  
that  
    /// you outline with a method like  
``stroke(_:with:style:)``, or of the  
    /// interior of a region that you  
fill with the ``fill(_:with:style:)``  
    /// method. Get a shading instance by  
calling one of the `Shading`  
    /// structure's factory methods. You  
can base shading on:  
    /// - A ``Color``.  
    /// - A ``Gradient``.  
    /// - Any type that conforms to  
``ShapeStyle``.
```

```
    /// - An ``Image``.  
    /// - What you've already drawn into  
the context.  
    /// - A collection of other shading  
instances.  
    public struct Shading : Sendable {  
  
        /// A shading instance that draws  
a copy of the current background.  
        public static var backdrop:  
GraphicsContext.Shading { get }  
  
        /// A shading instance that fills  
with the foreground style from  
        /// the graphics context's  
environment.  
        public static var foreground:  
GraphicsContext.Shading { get }  
  
        /// Returns a multilevel shading  
instance constructed from an  
        /// array of shading instances.  
        ///  
        /// - Parameter array: An array  
of shading instances. The array must  
        /// contain at least one  
element.  
        /// - Returns: A shading instance  
composed from the given instances.  
        public static func palette(_  
array: [GraphicsContext.Shading]) ->  
GraphicsContext.Shading
```

```
        /// Returns a shading instance  
        that fills with a color.  
        ///  
        /// - Parameter color: A  
``Color`` instance that defines the color  
        /// of the shading.  
        /// - Returns: A shading instance  
filled with a color.  
    public static func color(_ color:  
Color) -> GraphicsContext.Shading
```

```
        /// Returns a shading instance  
        that fills with a color in the given  
        /// color space.  
        ///  
        /// - Parameters:  
        /// - colorSpace: The RGB color  
space used to define the color. The  
        /// default is  
``Color/RGBColorSpace/sRGB``.  
        /// - red: The red component of  
the color.  
        /// - green: The green  
component of the color.  
        /// - blue: The blue component  
of the color.  
        /// - opacity: The opacity of  
the color. The default is `1`, which  
        /// means fully opaque.  
        /// - Returns: A shading instance  
filled with a color.  
    public static func color(_  
colorSpace: Color.RGBColorSpace = .sRGB,
```

```
red: Double, green: Double, blue: Double,  
opacity: Double = 1) ->  
GraphicsContext.Shading
```

```
    /// Returns a shading instance  
that fills with a monochrome color in  
    /// the given color space.  
    ///  
    /// - Parameters:  
    /// - colorSpace: The RGB color  
space used to define the color. The  
    /// default is  
``Color/RGBColorSpace/sRGB``.
```

```
    /// - white: The value to use  
for each of the red, green, and blue  
    /// components of the color.  
    /// - opacity: The opacity of  
the color. The default is `1`, which  
    /// means fully opaque.
```

```
    /// - Returns: A shading instance  
filled with a color.
```

```
public static func color(_  
colorSpace: Color.RGBColorSpace = .sRGB,  
white: Double, opacity: Double = 1) ->  
GraphicsContext.Shading
```

```
    /// Returns a shading instance  
that fills with the results of  
    /// querying a shader for each  
pixel.
```

```
    ///  
    /// For a shader function to act  
as a shape fill it must have a
```

```
    /// function signature matching:  
    ///  
    ///     [[ stitchable ]] half4  
name(float2 position, args...)  
    ///  
    /// where `position` is the user-  
space coordinates of the pixel applied  
    /// to the shader, and `args...`  
should be compatible with the uniform  
    /// arguments bound to `shader`.  
The function should return the  
    /// premultiplied color value in  
the color space of the destination  
    /// (typically sRGB).  
    ///  
    /// - Parameters:  
    ///     - shader: The shader  
defining the filled colors.  
    ///     - bounds: The rect used to  
define any `bounds` arguments  
    ///         of the shader.  
    ///  
    /// - Returns: A shading instance  
that fills using the shader.  
    @available(iOS 17.0, macOS 14.0,  
tvOS 17.0, *)  
    @available(watchOS, unavailable)  
    public static func shader(  
        shader: Shader, bounds: CGRect = .zero)  
    -> GraphicsContext.Shading  
  
    /// Returns a shading instance  
that fills with a mesh gradient.
```

```
    /**
     * - Parameters:
     *   - mesh: The mesh gradient defining the filled colors.
     */
    /**
     * - Returns: A shading that fills using the mesh gradient.
     */
    @available(iOS 18.0, macOS 15.0,
watchOS 11.0, tvOS 18.0, visionOS 2.0, *)
public static func meshGradient(_ mesh: MeshGradient) -> GraphicsContext.Shading
```

/// Returns a shading instance that fills with the given shape style.

```
        /**
         * Styles with geometry defined in a unit coordinate space
         * map that space to the rectangle associated with the drawn
         * object. You can adjust that using the ``ShapeStyle/in(_:)``
         * method. The shape style might affect the blend mode and opacity
         * of the drawn object.
         */
        /**
         * - Parameter style: A ``ShapeStyle`` instance to draw with.
         */
        /**
         * - Returns: A shading instance filled with a shape style.
         */
    public static func style<S>(_ style: S) -> GraphicsContext.Shading
    where S : ShapeStyle
```

```
    /// Returns a shading instance  
    that fills a linear (axial) gradient.  
    ///  
    /// The shading instance defines  
    an axis from `startPoint` to `endPoint`  
    /// in the current user space and  
    maps colors from `gradient`  
    /// to lines perpendicular to the  
    axis.  
    ///  
    /// - Parameters:  
    ///   - gradient: A ``Gradient``  
    instance that defines the colors  
    ///     of the gradient.  
    ///   - startPoint: The start  
    point of the gradient axis.  
    ///   - endPoint: The end point  
    of the gradient axis.  
    ///   - options: Options that you  
    use to configure the gradient.  
    /// - Returns: A shading instance  
    filled with a linear gradient.  
    public static func  
linearGradient(_ gradient: Gradient,  
startPoint: CGPoint, endPoint: CGPoint,  
options: GraphicsContext.GradientOptions  
= GradientOptions()) ->  
GraphicsContext.Shading  
  
    /// Returns a shading instance  
    that fills a radial gradient.  
    ///
```

```
    /// - Parameters:  
    ///   - gradient: A ``Gradient``  
instance that defines the colors  
        ///   of the gradient.  
    ///   - center: The point in the  
current user space on which SwiftUI  
        ///   centers the gradient.  
    ///   - startRadius: The distance  
from the center where the gradient  
        ///   starts.  
    ///   - endRadius: The distance  
from the center where the gradient ends.  
    ///   - options: Options that you  
use to configure the gradient.  
    /// - Returns: A shading instance  
filled with a radial gradient.  
public static func  
radialGradient(_ gradient: Gradient,  
center: CGPoint, startRadius: CGFloat,  
endRadius: CGFloat, options:  
GraphicsContext.GradientOptions =  
GradientOptions()) ->  
GraphicsContext.Shading  
  
    /// Returns a shading instance  
that fills a conic (angular) gradient.  
    ///  
    /// - Parameters:  
    ///   - gradient: A ``Gradient``  
instance that defines the colors  
        ///   of the gradient.  
    ///   - center: The point in the  
current user space on which SwiftUI
```

```
    /// centers the gradient.  
    /// - angle: The angle about  
the center that SwiftUI uses to start and  
    /// finish the gradient. The  
gradient sweeps all the way around the  
    /// center.  
    /// - options: Options that you  
use to configure the gradient.  
    /// - Returns: A shading instance  
filled with a conic gradient.  
public static func  
conicGradient(_ gradient: Gradient,  
center: CGPoint, angle: Angle = Angle(),  
options: GraphicsContext.GradientOptions  
= GradientOptions()) ->  
GraphicsContext.Shading
```

```
    /// Returns a shading instance  
that tiles an image across the infinite  
    /// plane.  
    ///  
    /// - Parameters:  
    /// - image: An ``Image`` to  
use as fill.  
    /// - origin: The point in the  
current user space where SwiftUI  
    /// places the bottom left  
corner of the part of the image  
    /// defined by `sourceRect`.  
The image repeats as needed.  
    /// - sourceRect: A unit space  
subregion of the image. The default  
    /// is a unit rectangle,
```

which selects the whole image.

    /// - scale: A factor that you can use to control the image size.

    /// - Returns: A shading instance filled with a tiled image.

```
    public static func tiledImage(_  
image: Image, origin: CGPoint = .zero,  
sourceRect: CGRect = CGRect(x: 0, y: 0,  
width: 1, height: 1), scale: CGFloat = 1)  
-> GraphicsContext.Shading  
}
```

    /// Options that affect the rendering of color gradients.

```
    ///  
    /// Use these options to affect how  
SwiftUI manages a gradient that you  
    /// create for a ``Shading`` instance  
for use in a ``GraphicsContext``.
```

```
    @frozen public struct GradientOptions  
: OptionSet {
```

        /// The corresponding value of the raw type.

```
        ///  
        /// A new instance initialized  
with `rawValue` will be equivalent to  
this
```

        /// instance. For example:

```
        ///
```

```
        ///     enum PaperSize: String {  
        ///         case A4, A5, Letter,
```

Legal

```
        /**
     */
    /**
     *      let selectedSize =
PaperSize.Letter
    /**
print(selectedSize.rawValue)
    /**
         // Prints "Letter"
    /**
    /**
         print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    /**
         // Prints "true"
public let rawValue: UInt32

        /**
Creates a new option set from
the given raw value.
    /**
        /**
This initializer always
succeeds, even if the value passed as
`rawValue`
        /**
exceeds the static properties
declared as part of the option set. This
        /**
example creates an instance
of `ShippingOptions` with a raw value
beyond
        /**
the highest element, with a
bit mask that effectively contains all
the
        /**
declared static members.
    /**
    /**
        let extraOptions =
ShippingOptions(rawValue: 255)
    /**

```

```
print(extraOptions.isStrictSuperset(of: .  
all))  
    ///      // Prints "true"  
    ///  
    /// - Parameter rawValue: The raw  
value of the option set to create. Each  
bit  
        /// of `rawValue` potentially  
represents an element of the option set,  
        /// though raw values may  
include bits that are not defined as  
distinct  
        /// values of the `OptionSet`  
type.  
    @inlinable public init(rawValue:  
UInt32)  
  
        /// An option that repeats the  
gradient outside its nominal range.  
        ///  
        /// Use this option to cause the  
gradient to repeat its pattern in  
        /// areas that exceed the bounds  
of its start and end points.  
        /// The repetitions use the same  
start and end value for each  
        /// repetition.  
        ///  
        /// Without this option or  
``mirror``, the gradient stops at  
        /// the end of its range. The  
``mirror`` option takes precedence if  
        /// you set both this one and
```

that one.

```
@inlinable public static var  
`repeat`: GraphicsContext.GradientOptions  
{ get }
```

```
    /// An option that repeats the  
gradient outside its nominal range,  
    /// reflecting every other  
instance.
```

```
    ///  
    /// Use this option to cause the  
gradient to repeat its pattern in  
    /// areas that exceed the bounds  
of its start and end points.
```

```
    /// The repetitions alternately  
reverse the start and end points,  
    /// producing a pattern like `0  
-> 1`, `1 -> 0`, `0 -> 1`, and so on.
```

```
    ///  
    /// Without either this option or  
``repeat``, the gradient stops at  
    /// the end of its range. This  
option takes precedence if  
    /// you set both this one and  
``repeat``.
```

```
@inlinable public static var  
mirror: GraphicsContext.GradientOptions {  
get }
```

```
    /// An option that interpolates  
between colors in a linear color space.
```

```
@inlinable public static var  
linearColor:
```

```
GraphicsContext.GradientOptions { get }

    /// The type of the elements of
    an array literal.
    @available(iOS 15.0, tvOS 15.0,
    watchOS 8.0, macOS 12.0, *)
    public typealias
    ArrayLiteralElement =
    GraphicsContext.GradientOptions

    /// The element type of the
    option set.
    /**
     * To inherit all the default
     * implementations from the `OptionSet`
     * protocol,
     * the `Element` type must be
     * `Self`, the default.
     * @available(iOS 15.0, tvOS 15.0,
     * watchOS 8.0, macOS 12.0, *)
     * public typealias Element =
     * GraphicsContext.GradientOptions

    /// The raw type that can be used
    to represent all values of the conforming
    /**
     * type.
     */
    /**
     * Every distinct value of the
     * conforming type has a corresponding
     * unique
     * value of the `RawValue` type,
     * but there may be values of the `RawValue`  

     * type that don't have a
```

corresponding value of the conforming type.

```
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
public typealias RawValue =
UInt32
}
```

    /// Returns a version of a shading resolved with the current values

    /// of the graphics context's environment.

    ///

    /// Calling this function once and then drawing multiple times with

    /// the result will often have less overhead than drawing with the

    /// original shading multiple times.

```
public func resolve(_ shading:
GraphicsContext.Shading) ->
GraphicsContext.Shading
```

    /// Draws a new layer, created by drawing code that you provide, into the

    /// context.

    ///

    /// – Parameter context: A closure that receives a new ``GraphicsContext``

    /// as input. This context represents a new transparency layer that you

    /// can draw into. When the closure returns, SwiftUI draws the new layer

```
    /// into the current context.  
    public func drawLayer(content: (inout  
    GraphicsContext) throws -> Void) rethrows  
  
        /// Draws a path into the context and  
        fills the outlined region.  
        ///  
        /// The current drawing state of the  
        context defines the  
        /// full drawing operation. For  
        example, the current transformation and  
        /// clip shapes, and any styles  
        applied to the result, affect the final  
        /// result.  
        ///  
        /// - Parameters:  
        ///     - path: The outline of the  
        region to fill.  
        ///     - shading: The color or pattern  
        to use when filling the region  
        ///     bounded by `path`.  
        ///     - style: A style that indicates  
        how to rasterize the path.  
        public func fill(_ path: Path, with  
        shading: GraphicsContext.Shading, style:  
        FillStyle = FillStyle())  
  
        /// Draws a path into the context  
        with a specified stroke style.  
        ///  
        /// If you only need to control the  
        style's ``StrokeStyle/lineWidth``  
        /// property, use
```

```
``stroke(_:with:lineWidth:)`` instead.  
///  
/// - Parameters:  
///   - path: The path to outline.  
///   - shading: The color or pattern  
to use when outlining the `path`.  
///   - style: A style that indicates  
how to outline the path.  
public func stroke(_ path: Path, with  
shading: GraphicsContext.Shading, style:  
StrokeStyle)
```

```
/// Draws a path into the context  
with a specified line width.  
///  
/// When you call this method, all  
``StrokeStyle`` properties other than  
/// ``StrokeStyle/lineWidth`` take  
their default values. To control other  
/// style properties, use  
``stroke(_:with:style:)`` instead.  
///  
/// - Parameters:  
///   - path: The path to outline.  
///   - shading: The color or pattern  
to use when outlining the `path`.  
///   - lineWidth: The width of the  
stroke, which defaults to `1`.  
public func stroke(_ path: Path, with  
shading: GraphicsContext.Shading,  
lineWidth: CGFloat = 1)
```

```
/// An image resolved to a particular
```

environment.

```
///  
/// You resolve an ``Image`` in  
preparation for drawing it into a  
context,  
/// either manually by calling  
``resolve(_:)898z6``, or automatically  
/// when calling ``draw(_:in:style:)‑  
blhz`` or ``draw(_:at:anchor:)‑1z5wt``.  
/// The resolved image takes into  
account environment values like the  
/// display resolution and current  
color scheme.  
public struct ResolvedImage {  
  
    /// The size of the image.  
public var size: CGSize { get }  
  
    /// The distance from the top of  
the image to its baseline.  
    ///  
    /// If the image has no baseline,  
this value is equivalent to the  
    /// image's height.  
public let baseline: CGFloat  
  
    /// An optional shading to fill  
the image with.  
    ///  
    /// The value of this property  
defaults to  
    ///  
``GraphicsContext/Shading/foreground``
```

```
for template images, and
    /// to `nil` otherwise.
    public var shading:
GraphicsContext.Shading?
}

    /// Gets a version of an image that's
fixed with the current values of
    /// the graphics context's
environment.
    ///
    /// You can measure the resolved
image by looking at its
    /// ``ResolvedImage/size`` and
``ResolvedImage/baseline`` properties.
    /// You can draw the resolved image
with the context's
    /// ``draw(_:_in:style:)`` or
``draw(_:_at:anchor:)`` method.
    ///
    /// - Parameter image: The ``Image``
to resolve.
    /// - Returns: An image that's
resolved into the current context's
    /// environment, taking into
account environment values like the
    /// display resolution and current
color scheme.
public func resolve(_ image: Image)
-> GraphicsContext.ResolvedImage

    /// Draws a resolved image into the
context, using the specified rectangle
```

```
    /// as a layout frame.  
    ///  
    /// The current context state defines  
    the full drawing operation. For  
    /// example, the current  
    transformation and clip shapes affect how  
    SwiftUI  
    /// draws the image.  
    ///  
    /// - Parameters:  
    ///   - image: The ``ResolvedImage``  
    to draw. Get a resolved image from an  
    ///   ``Image`` by calling  
    ``resolve(_:)``. Alternatively, you  
    can  
    ///   call ``draw(_:in:style:)``-  
    blhz`` with an ``Image``, and that method  
    ///   performs the resolution  
    automatically.  
    ///   - rect: The rectangle in the  
    current user space to draw the image in.  
    ///   - style: A fill style to use  
    when rasterizing the image.  
    public func draw(_ image:  
    GraphicsContext.ResolvedImage, in rect:  
    CGRect, style: FillStyle = FillStyle())  
  
    /// Draws a resolved image into the  
    context, aligning an anchor within the  
    /// image to a point in the context.  
    ///  
    /// The current context state defines  
    the full drawing operation. For
```

```
    /// example, the current
    transformation and clip shapes affect how
SwiftUI
    /// draws the image.
    ///
    /// - Parameters:
    ///   - image: The ``ResolvedImage`` to draw. Get a resolved image from an
    ///     ``Image`` by calling ``resolve(_:)``-898z6``. Alternatively, you can
    ///     call ``draw(_:at:anchor:)``-7l217`` with an ``Image``, and that method
    ///     performs the resolution automatically.
    ///   - point: A point within the rectangle of the resolved image to anchor
    ///     to a point in the context.
    ///   - anchor: A ``UnitPoint`` within the context to align the image with.
    ///     The default is ``UnitPoint/center``.
public func draw(_ image:
GraphicsContext.ResolvedImage, at point:
CGPoint, anchor: UnitPoint = .center)

    /// Draws an image into the context, using the specified rectangle
    /// as a layout frame.
    ///
    /// The current context state defines
```

the full drawing operation. For  
    /// example, the current  
transformation and clip shapes affect how  
SwiftUI

    /// draws the image.  
    ///  
    /// - Parameters:  
    /// - image: The ``Image`` to draw.  
Before drawing, the method converts  
    /// the image to a  
``ResolvedImage`` by calling  
``resolve(\_:)``-898z6``.  
    /// - rect: The rectangle in the  
current user space to draw the image in.  
    /// - style: A fill style to use  
when rasterizing the image.

```
public func draw(_ image: Image, in  
rect: CGRect, style: FillStyle =  
FillStyle())
```

    /// Draws an image into the context,  
aligning an anchor within the image  
    /// to a point in the context.  
    ///  
    /// The current context state defines  
the full drawing operation. For  
    /// example, the current  
transformation and clip shapes affect how  
SwiftUI

    /// draws the image.  
    ///  
    /// - Parameters:  
    /// - image: The ``Image`` to draw.

Before drawing, the method converts

```
    /// the image to a
``ResolvedImage`` by calling
``resolve(_:)``-898z6``.
    /// - point: A point within the
rectangle of the resolved image to anchor
    /// to a point in the context.
    /// - anchor: A ``UnitPoint``
within the context to align the image
with.
    /// The default is
``UnitPoint/center``.
public func draw(_ image: Image, at
point: CGPoint, anchor: UnitPoint
= .center)
```

/// A text view resolved to a particular environment.

```
///
/// You resolve a ``Text`` view in preparation for drawing it into a context,
    /// either manually by calling
``resolve(_:)``-4dx65`` or automatically
    /// when calling
``draw(_:in:)``-5opqf`` or
``draw(_:at:anchor:)``-5dgmd``.
    /// The resolved text view takes into account environment values like the
    /// display resolution and current color scheme.
```

```
public struct ResolvedText {
```

```
    /// The shading to fill uncolored
    text regions with.
    /**
     * This value defaults to the
     ``GraphicsContext/Shading/foreground``'
     * shading.
    public var shading:
GraphicsContext.Shading

    /// Measures the size of the
resolved text for a given
    /// area into which the text
should be placed.
    /**
     * - Parameter size: The area to
place the ``Text`` view in.
    public func measure(in size:
CGSize) -> CGSize

    /// Gets the distance from the
first line's ascender to its baseline.
    public func firstBaseline(in
size: CGSize) -> CGFloat

    /// Gets the distance from the
first line's ascender to the last
    /// line's baseline.
    public func lastBaseline(in size:
CGSize) -> CGFloat
}

    /// Gets a version of a text view
that's fixed with the current values of
```

```
    /// the graphics context's
    environment.
    ///
    /// You can measure the resolved text
    by calling its
    /// ``ResolvedText/measure(in:)`` method.
    /// You can draw the resolved text
    with the context's
    /// ``draw(_:in:)`` or
    ``draw(_:at:anchor:)`` method.
    ///
    /// - Parameter text: The ``Text``
    view to resolve.
    /// - Returns: A text view that's
    resolved into the current context's
    /// environment, taking into
    account environment values like the
    /// display resolution and current
    color scheme.
public func resolve(_ text: Text) ->
GraphicsContext.ResolvedText
```

```
    /// Draws resolved text into the
    context using the specified rectangle
    /// as a layout frame.
    ///
    /// The current context state defines
    the full drawing operation. For
    /// example, the current
    transformation and clip shapes affect how
    SwiftUI
    /// draws the text.
```

```
///  
/// - Parameters:  
///   - text: The ``ResolvedText`` to  
draw. Get resolved text from a  
///   ``Text`` view by calling  
``resolve(_:)``-4dx65``. Alternatively, you  
///   can call  
``draw(_:in:)``-5opqf`` with a ``Text``  
view, and that  
///   method performs the  
resolution automatically.  
///   - rect: The rectangle in the  
current user space to draw the text in.  
public func draw(text:  
GraphicsContext.ResolvedText, in rect:  
CGRect)
```

```
/// Draws resolved text into the  
context, aligning an anchor within the  
/// ideal size of the text to a point  
in the context.  
///  
/// The current context state defines  
the full drawing operation. For  
/// example, the current  
transformation and clip shapes affect how  
SwiftUI  
/// draws the text.  
///  
/// - Parameters:  
///   - text: The ``ResolvedText`` to  
draw. Get resolved text from a  
///   ``Text`` view by calling
```

```
``resolve(_:) - 4dx65``. Alternatively, you
    ///      can call
``draw(_:at:anchor:) - 5dgmd`` with a
``Text`` view, and that
    ///      method performs the
resolution automatically.
```

```
    /// - point: A point within the
rectangle of the ideal size of the
    ///      resolved text to anchor to a
point in the context.
```

```
    /// - anchor: A ``UnitPoint``
within the context to align the text
with.
```

```
    ///      The default is
``UnitPoint/center``.
```

```
public func draw(_ text:
GraphicsContext.ResolvedText, at point:
CGPoint, anchor: UnitPoint = .center)
```

```
    /// Draws text into the context using
the specified rectangle
    /// as a layout frame.
```

```
    ///
    /// The current context state defines
the full drawing operation. For
    /// example, the current
transformation and clip shapes affect how
SwiftUI
```

```
    /// draws the text.
```

```
    ///
    /// - Parameters:
```

```
    /// - text: The ``Text`` view to
draw. Before drawing, the method converts
```

```
    ///      the view to ``ResolvedText``  
by calling ``resolve(_:) -4dx65``.  
    /// - rect: The rectangle in the  
current user space to draw the text in.  
    public func draw(_ text: Text, in  
rect: CGRect)  
  
        /// Draws text into the context,  
aligning an anchor within the ideal size  
        /// of the rendered text to a point  
in the context.  
        ///  
        /// The current context state defines  
the full drawing operation. For  
        /// example, the current  
transformation and clip shapes affect how  
SwiftUI  
        /// draws the text.  
        ///  
        /// - Parameters:  
        /// - text: The ``Text`` view to  
draw. Before drawing, the method converts  
        ///      the view to ``ResolvedText``  
by calling ``resolve(_:) -4dx65``.  
        /// - point: A point within the  
rectangle of the resolved text to anchor  
        ///      to a point in the context.  
        /// - anchor: A ``UnitPoint``  
within the context to align the text  
with.  
        ///      The default is  
``UnitPoint/center``.  
    public func draw(_ text: Text, at
```

```
point: CGPoint, anchor: UnitPoint
= .center)

    /// A static sequence of drawing
operations that may be drawn
    /// multiple times, preserving their
resolution independence.
    ///
    /// You resolve a child view in
preparation for drawing it into a context
    /// by calling
``resolveSymbol(id:)``. The resolved view
takes into account
    /// environment values like the
display resolution and current color
scheme.

public struct ResolvedSymbol {

    /// The dimensions of the
resolved symbol.
    public var size: CGSize { get }

}

    /// Gets the identified child view as
a resolved symbol, if the view exists.
    ///
    /// - Parameter id: The value that
you used to tag the view when you
    /// define it in the `symbols`
parameter of the ``Canvas`` initializer
    ///
``Canvas/init(opaque:colorMode:rendersAsy
nchronously:renderer:symbols:)``.
```

```
    /// - Returns: The resolved symbol,  
    or `nil` if SwiftUI can't find a child  
    /// view with the given `id`.  
    public func resolveSymbol<ID>(id: ID)  
-> GraphicsContext.ResolvedSymbol? where  
ID : Hashable
```

```
    /// Draws a resolved symbol into the  
    context, using the specified rectangle  
    /// as a layout frame.  
    ///  
    /// The current context state defines  
    the full drawing operation. For  
    /// example, the current  
    transformation and clip shapes affect how  
    SwiftUI  
    /// draws the symbol.  
    ///  
    /// - Parameters:  
    ///     - symbol: The  
    ``ResolvedSymbol`` to draw. Get a  
    resolved symbol  
    ///     by calling  
    ``resolveSymbol(id:)`` with the  
    identifier that you  
    ///     use to tag the corresponding  
    child view during ``Canvas``  
    ///     initialization.  
    ///     - rect: The rectangle in the  
    current user space to draw the symbol in.  
    public func draw(_ symbol:  
GraphicsContext.ResolvedSymbol, in rect:  
CGRect)
```

```
    /// Draws a resolved symbol into the
    context, aligning an anchor within the
    /// symbol to a point in the context.
    ///
    /// The current context state defines
    the full drawing operation. For
    /// example, the current
    transformation and clip shapes affect how
    SwiftUI
    /// draws the symbol.
    ///
    /// - Parameters:
    ///   - symbol: The
    ``ResolvedSymbol`` view to draw. Get a
    resolved symbol
    /// by calling
    ``resolveSymbol(id:)`` with the
    identifier that you
    /// use to tag the corresponding
    child view during ``Canvas``
    /// initialization.
    /// - point: A point within the
    rectangle of the resolved symbol to
    anchor
    /// to a point in the context.
    /// - anchor: A ``UnitPoint``
    within the context to align the symbol
    with.
    /// The default is
    ``UnitPoint/center``.
public func draw(_ symbol:
GraphicsContext.ResolvedSymbol, at point:
```

```
CGPoint, anchor: UnitPoint = .center)
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension GraphicsContext {

    /// Provides a Core Graphics context
    that you can use as a proxy to draw
    /// into this context.
    ///
    /// Use this method to use existing
    drawing code that relies on
    /// Core Graphics primitives.
    ///
    /// - Parameter content: A closure
    that receives a
    ///
<doc://com.apple.documentation/documentation/CoreGraphics/CGContext>
    /// that you use to perform drawing
    operations, just like you draw into a
    /// ``GraphicsContext`` instance.
Any filters, blend mode settings, clip
    /// masks, and other state set
before calling `withCGContext(content:)`
    /// apply to drawing operations in
the Core Graphics context as well. Any
    /// state you set on the Core
Graphics context is lost when the closure
    /// returns. Accessing the Core
Graphics context after the closure
    /// returns produces undefined
```

behavior.

```
    public func withCGContext(content:  
        (CGContext) throws -> Void) rethrows  
}
```

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

```
    /// Draws `line` into the graphics  
    context.
```

```
    public func draw(_ line:  
        Text.Layout.Line, options:  
        Text.Layout.DrawingOptions = .init())
```

```
    /// Draws `run` into the graphics  
    context.
```

```
    public func draw(_ run:  
        Text.Layout.Run, options:  
        Text.Layout.DrawingOptions = .init())
```

```
    /// Draws `slice` into the graphics  
    context.
```

```
    public func draw(_ slice:  
        Text.Layout.RunSlice, options:  
        Text.Layout.DrawingOptions = .init())  
}
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension GraphicsContext.BlendMode :  
BitwiseCopyable {
```

```
}
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension GraphicsContext.BlendMode :  
Sendable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension GraphicsContext.ClipOptions :  
Sendable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension GraphicsContext.ClipOptions :  
BitwiseCopyable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension GraphicsContext.ShadowOptions :  
Sendable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension GraphicsContext.ShadowOptions :  
BitwiseCopyable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)
```

```
extension GraphicsContext.BlurOptions :  
Sendable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension GraphicsContext.BlurOptions :  
BitwiseCopyable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension GraphicsContext.FilterOptions :  
Sendable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension GraphicsContext.FilterOptions :  
BitwiseCopyable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension GraphicsContext.GradientOptions  
: Sendable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension GraphicsContext.GradientOptions  
: BitwiseCopyable {  
}
```

```
/// A type that collects multiple
instances of a content type --- like
views,
/// scenes, or commands --- into a single
unit.
///
/// Use a group to collect multiple views
into a single instance, without
/// affecting the layout of those views,
like an ``SwiftUI/HStack``,
/// ``SwiftUI/VStack``, or
``SwiftUI/Section`` would. After creating
a group,
/// any modifier you apply to the group
affects all of that group's members.
/// For example, the following code
applies the ``SwiftUI/Font/headline``
/// font to three views in a group.
///
///     Group {
///         Text("SwiftUI")
///         Text("Combine")
///         Text("Swift System")
///     }
///     .font(.headline)

/// Because you create a group of views
with a ``SwiftUI/ViewBuilder``, you can
/// use the group's initializer to
produce different kinds of views from a
/// conditional, and then optionally
apply modifiers to them. The following
```

```
/// example uses a `Group` to add a
/// navigation bar title,
/// regardless of the type of view the
conditional produces:
///
///     Group {
///         if isLoggedIn {
///             WelcomeView()
///         } else {
///             LoginView()
///         }
///     }
///     .navigationBarTitle("Start")
///

/// The modifier applies to all members
of the group --- and not to the group
/// itself. For example, if you apply
``View/onAppear(perform:)`` to the above
/// group, it applies to all of the views
produced by the `if isLoggedIn`
/// conditional, and it executes every
time `isLoggedIn` changes.
///
/// Because a group of views itself is a
view, you can compose a group within
/// other view builders, including
nesting within other groups. This allows
you
/// to add large numbers of views to
different view builder containers. The
/// following example uses a `Group` to
collect 10 ``SwiftUI/Text`` instances,
/// meaning that the vertical stack's
```

```
view builder returns only two views ---  
/// the group, plus an additional  
``SwiftUI/Text``:  
///  
///     var body: some View {  
///         VStack {  
///             Group {  
///                 Text("1")  
///                 Text("2")  
///                 Text("3")  
///                 Text("4")  
///                 Text("5")  
///                 Text("6")  
///                 Text("7")  
///                 Text("8")  
///                 Text("9")  
///                 Text("10")  
///             }  
///             Text("11")  
///         }  
///     }  
  
/// You can initialize groups with  
several types other than  
``SwiftUI/View``,  
/// such as ``SwiftUI/Scene`` and  
``SwiftUI/ToolbarContent``. The closure  
you  
/// provide to the group initializer uses  
the corresponding builder type  
/// (``SwiftUI/SceneBuilder``,  
``SwiftUI/ToolbarContentBuilder``), and so  
on),
```

```
/// and the capabilities of these
builders vary between types. For example,
/// you can use groups to return large
numbers of scenes or toolbar content
/// instances, but not to return
different scenes or toolbar content based
/// on conditionals.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct Group<Content> {

    /// The type of view representing the
body of this view.
    ///
    /// When you create a custom view,
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
    public typealias Body = Never
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Group : View where Content : View {

    /// Creates a group of views.
    /// - Parameter content: A
``SwiftUI/ViewBuilder`` that produces the
views
    /// to group.
    @inlinable nonisolated public
init(@ViewBuilder content: () -> Content)
```

}

```
///  
SecondaryCard { subview[1] }  
///  
///  
if let  
first = subviews.first {  
    ///  
    FeatureCard { first }  
    ///  
    ///  
    if  
subviews.count >= 3 {  
    ///  
    SecondaryCard { subviews[2] }  
    ///  
    ///  
    ///  
    if  
subviews.count > 3 {  
    ///  
    subviews[3...]  
    ///  
    ///  
    ///  
    ///  
    ///  
    ///  
    ///  
    ///  
    /// You can use `CardsView` with its  
view builder-based initializer to  
/// arrange a collection of subviews:  
///  
///     CardsView {  
///         NavigationLink("What's  
New!") { WhatsNewView() }  
///         NavigationLink("Latest  
Hits") { LatestHitsView() }
```

```
///  
NavigationLink("Favorites")  
{ FavoritesView() }  
///  
NavigationLink("Playlists")  
{ MyPlaylists() }  
///  
/// Subviews collection constructs  
subviews on demand, so only access the  
/// part of the collection you need  
to create the resulting content.  
///  
/// Subviews are proxies to the view  
they represent, which means  
/// that modifiers that you apply to  
the original view take effect before  
/// modifiers that you apply to the  
Subview. SwiftUI resolves the view  
/// using the environment of its  
container rather than the environment of  
/// its subview proxy. Additionally,  
because subviews represent a  
/// single view or container, a  
Subview might represent a view after the  
/// application of styles. As a  
result, applying a style to a subview  
might  
/// have no effect.  
///  
/// - Parameters:  
///     - view: The view to get the  
subviews of.
```

```
    /// - transform: A closure that
    constructs a view from the collection of
    ///     subviews.
    public init<Base, Result>(subviews
view: Base, @ViewBuilder transform:
@escaping (SubviewsCollection) -> Result)
where Content ==
GroupElementsOfContent<Base, Result>,
Base : View, Result : View
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Group {

    /// Constructs a group from the
    sections of the given view.
    ///
    /// Sections are constructed lazily,
    on demand, so access only as much
    /// of this collection as is
    necessary to create the resulting
    content.
    ///
    ///     struct
SectionedStack<Content: View>: View {
    ///         var content: Content
    ///
    ///             init(@ViewBuilder
content: () -> Content) {
    ///                 self.content =
content()
    ///             }
}
```



```
    ///             Text("World")
    ///         } footer: {
    ///             Text("Footer A")
    ///         }
    ///     Section("Header B") {
    ///         Text("Foo")
    ///         Text("Bar")
    ///     } footer: {
    ///         Text("Footer B")
    ///     }
    /// }
    ///
    /// Any content of the given view
    which is not explicitly specified as a
    /// section is grouped with its
    sibling content to form implicit
    sections,
    /// meaning the minimum number of
    sections in a `SectionCollection` is one.
    /// For example in the following
    `SectionedStack`, there is one explicit
    /// section, and two implicit
    sections containing the content before,
    /// and after the explicit section:
    ///
    ///     SectionedStack {
    ///         Text("First implicit
section")
    ///         Section("Explicit
section") {
    ///             Text("Content")
    ///         }
    ///     Text("Second implicit
```

```
section")
    /**
     */
    /**
     * - Parameters:
     *   - view: The view to extract the
     * sections of.
     *   - content: A closure that
     * constructs a view from the collection of
     *   sections.
    public init<Base, Result>(sections
view: Base, @ViewBuilder transform:
@escaping (SectionCollection) -> Result)
where Content ==
GroupSectionsOfContent<Base, Result>,
Base : View, Result : View
}
```

```
/// Transforms the subviews of a given
view into a resulting content view.
///
/// You don't use this type directly.
Instead SwiftUI creates this type on
/// your behalf.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct
GroupElementsOfContent<Subviews, Content>
: View where Subviews : View, Content :
View {

    /// The content and behavior of the
view.
    ///
```

```
    /// When you implement a custom view,  
you must implement a computed  
    /// `body` property to provide the  
content for your view. Return a view  
    /// that's composed of built-in views  
that SwiftUI provides, plus other  
    /// composite views that you've  
already defined:  
    ///  
    ///     struct MyView: View {  
    ///         var body: some View {  
    ///             Text("Hello, World!")  
    ///         }  
    ///     }  
    ///  
    /// For more information about  
composing views and a view hierarchy,  
    /// see <doc:Declaring-a-Custom-  
View>.
```

```
    @MainActor @preconcurrency public var  
body: some View { get }  
  
    /// The type of view representing the  
body of this view.  
    ///  
    /// When you create a custom view,  
Swift infers this type from your  
    /// implementation of the required  
``View/body-swift.property`` property.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias Body = some View
```

```
}
```

```
/// Transforms the sections of a given
view into a resulting content view.
///
/// You don't use this type directly.
Instead SwiftUI creates this type on
/// your behalf.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct
GroupSectionsOfContent<Sections, Content>
: View where Sections : View, Content : View {

    /// The content and behavior of the
view.
    ///
    /// When you implement a custom view,
you must implement a computed
    /// `body` property to provide the
content for your view. Return a view
    /// that's composed of built-in views
that SwiftUI provides, plus other
    /// composite views that you've
already defined:
    ///
    ///     struct MyView: View {
    ///         var body: some View {
    ///             Text("Hello, World!")
    ///         }
    ///     }
    ///
}
```

```
    /// For more information about
    /// composing views and a view hierarchy,
    /// see <doc:Declaring-a-Custom-
View>.
    @MainActor @preconcurrency public var
body: some View { get }

    /// The type of view representing the
body of this view.
    ///
    /// When you create a custom view,
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias Body = some View
}

/// A view that arranges its subviews in
a horizontal line.
///
/// Unlike ``LazyHStack``, which only
renders the views when your app needs to
/// display them onscreen, an `HStack`
renders the views all at once, regardless
/// of whether they are on- or offscreen.
Use the regular `HStack` when you have
/// a small number of subviews or don't
want the delayed rendering behavior
/// of the "lazy" version.
///
```

```
/// The following example shows a simple
/// horizontal stack of five text views:
///
///     var body: some View {
///         HStack(
///             alignment: .top,
///             spacing: 10
///         ) {
///             ForEach(
///                 1...5,
///                 id: \.self
///             ) {
///                 Text("Item \($0)")
///             }
///         }
///     }
///
/// ! [Five text views, named Item 1
/// through Item 5, arranged in a
/// horizontal row.] (SwiftUI-HStack-
/// simple.png)
///
/// > Note: If you need a horizontal
/// stack that conforms to the ``Layout``
/// protocol, like when you want to
/// create a conditional layout using
/// ``AnyLayout``, use ``HStackLayout``
/// instead.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct HStack<Content> : View where Content : View {
```

```
    /// Creates a horizontal stack with
    the given spacing and vertical alignment.
    ///
    /// - Parameters:
    ///   - alignment: The guide for
    aligning the subviews in this stack. This
    ///   guide has the same vertical
    screen coordinate for every subview.
    ///   - spacing: The distance between
    adjacent subviews, or `nil` if you
    ///   want the stack to choose a
    default distance for each pair of
    ///   subviews.
    ///   - content: A view builder that
    creates the content of this stack.
    @inlinable public init(alignment:
VerticalAlignment = .center, spacing:
CGFloat? = nil, @ViewBuilder content: () -> Content)
```

```
    /// The type of view representing the
body of this view.
    ///
    /// When you create a custom view,
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Body = Never
}
```

```
/// A horizontal container that you can
```

```
use in conditional layouts.

///  
/// This layout container behaves like an  
``HStack``, but conforms to the  
/// ``Layout`` protocol so you can use it  
in the conditional layouts that you  
/// construct with ``AnyLayout``. If you  
don't need a conditional layout, use  
/// ``HStack`` instead.  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
@frozen public struct HStackLayout :  
Layout {  
  
    /// The vertical alignment of  
    subviews.  
    public var alignment:  
    VerticalAlignment  
  
    /// The distance between adjacent  
    subviews.  
    ///  
    /// Set this value to `nil` to use  
    default distances between subviews.  
    public var spacing: CGFloat?  
  
    /// Creates a horizontal stack with  
    the specified spacing and vertical  
    /// alignment.  
    ///  
    /// - Parameters:  
    ///     - alignment: The guide for  
    aligning the subviews in this stack. It
```

```
    ///      has the same vertical
    screen coordinate for all subviews.
    ///      - spacing: The distance
    between adjacent subviews. Set this value
    ///      to `nil` to use default
    distances between subviews.
    @inlinable public init(alignment:
VerticalAlignment = .center, spacing:
CGFloat? = nil)

    /// The type defining the data to
animate.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias AnimatableData =
EmptyAnimatableData

    /// Cached values associated with the
layout instance.
    ///
    /// If you create a cache for your
custom layout, you can use
    /// a type alias to define this type
as your data storage type.
    /// Alternatively, you can refer to
the data storage type directly in all
    /// the places where you work with
the cache.
    ///
    /// See ``makeCache(subviews:)`` for
more information.
    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
```

```
    public typealias Cache
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension HStackLayout : Sendable {
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension HStackLayout : BitwiseCopyable
{
}

/// A shape style that maps to one of the
/// numbered content styles.
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
@frozen public struct
HierarchicalShapeStyle : ShapeStyle {

    /// A shape style that maps to the
    /// first level of the current
    /// content style.
    public static let primary:
HierarchicalShapeStyle

    /// A shape style that maps to the
    /// second level of the current
    /// content style.
    public static let secondary:
HierarchicalShapeStyle
```

```
    /// A shape style that maps to the
    third level of the current
    /// content style.
    public static let tertiary:
HierarchicalShapeStyle

    /// A shape style that maps to the
    fourth level of the current
    /// content style.
    public static let quaternary:
HierarchicalShapeStyle

    /// The type of shape style this will
    resolve to.
    ///
    /// When you create a custom shape
    style, Swift infers this type
    /// from your implementation of the
    required `resolve` function.
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Resolved = Never
}

@available(iOS 16.0, macOS 12.0,
macCatalyst 15.0, tvOS 17.0, watchOS
10.0, *)
extension HierarchicalShapeStyle {

    /// A shape style that maps to the
    fifth level of the current
    /// content style.
    public static let quinary:
```

```
HierarchicalShapeStyle
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension HierarchicalShapeStyle : BitwiseCopyable {

/// Styles that you can apply to hierarchical shapes.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@frozen public struct HierarchicalShapeStyleModifier<Base> : ShapeStyle where Base : ShapeStyle {

    /// The type of shape style this will resolve to.
    ///
    /// When you create a custom shape style, Swift infers this type
    /// from your implementation of the required `resolve` function.
    @available(iOS 17.0, tvOS 17.0,
    watchOS 10.0, macOS 14.0, *)
    public typealias Resolved = Never
}

/// An alignment position along the horizontal axis.
///
/// Use horizontal alignment guides to
```

```
tell SwiftUI how to position views
/// relative to one another horizontally,
like when you place views vertically
/// in an ``VStack``. The following
example demonstrates common built-in
/// horizontal alignments:
///
/// ! [Three columns of content. Each
column contains a string
/// inside a box with a vertical line
above and below the box. The
/// lines are aligned horizontally with
the text in a different way for each
/// column. The lines for the left-most
string, labeled Leading, align with
/// the left edge of the string. The
lines for the middle string, labeled
/// Center, align with the center of the
string. The lines for the right-most
/// string, labeled Trailing, align with
the right edge of the
/// string.] (HorizontalAlignment-1-iOS)
///
/// You can generate the example above by
creating a series of columns
/// implemented as vertical stacks, where
you configure each stack with a
/// different alignment guide:
///
///     private struct
HorizontalAlignmentGallery: View {
    ///         var body: some View {
    ///             HStack(spacing: 30) {
```

```
///  
column(alignment: .leading, text:  
"Leading")  
///  
column(alignment: .center, text:  
"Center")  
///  
column(alignment: .trailing, text:  
"Trailing")  
/// }  
/// .frame(height: 150)  
/// }  
///  
/// private func  
column(alignment: HorizontalAlignment,  
text: String) -> some View {  
/// VStack(alignment:  
alignment, spacing: 0) {  
///  
Color.red.frame(width: 1)  
///  
Text(text).font(.title).border(.gray)  
///  
Color.red.frame(width: 1)  
/// }  
/// }  
///  
/// During layout, SwiftUI aligns the  
views inside each stack by bringing  
/// together the specified guides of the  
affected views. SwiftUI calculates  
/// the position of a guide for a
```

particular view based on the characteristics

/// of the view. For example, the ``HorizontalAlignment/center`` guide appears

/// at half the width of the view. You can override the guide calculation for a particular view using the ``View/alignmentGuide(\_:computeValue:)`` view modifier.

///

/// **Layout direction**

///

/// When a user configures their device to use a left-to-right language like English, the system places the leading alignment on the left and the trailing alignment on the right, as the example from the previous section demonstrates. However, in a right-to-left language, the system reverses these. You can see this by using the ``View/environment(\_:\_:)`` view modifier to explicitly override the ``EnvironmentValues/layoutDirection`` environment value for the view defined above:

///

///     HorizontalAlignmentGallery()  
///         .environment(\.layoutDirection, .rightToLeft)

///

/// ! [Three columns of content. Each

column contains a string  
/// inside a box with a vertical line  
above and below the box. The  
/// lines are aligned horizontally with  
the text in a different way for each  
/// column. The lines for the left-most  
string, labeled Trailing, align with  
/// the left edge of the string. The  
lines for the middle string, labeled  
/// Center, align with the center of the  
string. The lines for the right-most  
/// string, labeled Leading, align with  
the right edge of the  
/// string.] (HorizontalAlignment-2-iOS)  
///  
/// This automatic layout adjustment  
makes it easier to localize your app,  
/// but it's still important to test your  
app for the different locales that  
/// you ship into. For more information  
about the localization process, see  
///  
[<doc://com.apple.documentation/documentation/Xcode/localization>](doc://com.apple.documentation/documentation/Xcode/localization).  
///  
/// **Custom alignment guides**  
///  
/// You can create a custom horizontal  
alignment by creating a type that  
/// conforms to the ``AlignmentID``  
protocol, and then using that type to  
/// initialize a new static property on  
`HorizontalAlignment`:

```
///  
///     private struct  
OneQuarterAlignment: AlignmentID {  
///         static func defaultValue(in  
context: ViewDimensions) -> CGFloat {  
///             context.width / 4  
///         }  
///     }  
///  
///     extension HorizontalAlignment {  
///         static let oneQuarter =  
HorizontalAlignment(OneQuarterAlignment.s  
elf)  
///     }  
///  
/// You implement the  
``AlignmentID/defaultValue(in:)`` method  
to calculate  
/// a default value for the custom  
alignment guide. The method receives a  
/// ``ViewDimensions`` instance that you  
can use to calculate an appropriate  
/// value based on characteristics of the  
view. The example above places  
/// the guide at one quarter of the width  
of the view, as measured from the  
/// view's origin.  
///  
/// You can then use the custom alignment  
guide like any built-in guide. For  
/// example, you can use it as the  
`alignment` parameter to a ``VStack``,  
/// or you can change it for a specific
```

```
view using the
///
``View/alignmentGuide(_:computeValue:)``
view modifier.
/// Custom alignment guides also
/// automatically reverse in a right-to-left
/// environment, just like built-in
/// guides.
///
/// Composite alignment
///
/// Combine a ``VerticalAlignment`` with
/// a `HorizontalAlignment` to create a
/// composite ``Alignment`` that
/// indicates both vertical and horizontal
/// positioning in one value. For
/// example, you could combine your custom
/// `oneQuarter` horizontal alignment
/// from the previous section with a built-in
/// ``VerticalAlignment/center`` vertical
/// alignment to use in a ``ZStack``:
///
///     struct LayeredVerticalStripes:
View {
    ///         var body: some View {
    ///             ZStack(alignment:
    Alignment(horizontal: .oneQuarter,
    vertical: .center)) {
    ///
    verticalStripes(color: .blue)
    ///                         .frame(width:
    300, height: 150)
    ///
```

```
verticalStripes(color: .green)
///                                .frame(width:
180, height: 80)
///                                }
///                                }
///                                }
///                                private func
verticalStripes(color: Color) -> some
View {
///                                HStack(spacing: 1) {
///                                ForEach(0..<4) { _ in
color }
///                                }
///                                }
///                                }
///                                }

/// The example above uses widths and
heights that generate two mismatched sets
/// of four vertical stripes. The
``ZStack`` centers the two sets
vertically and
/// aligns them horizontally one quarter
of the way from the leading edge of
/// each set. In a left-to-right locale,
this aligns the right edges of the
/// left-most stripes of each set:
///
/// ! [Two sets of four rectangles. The
first set is blue. The
/// second set is green, is smaller, and
is layered on top of the first set.
/// The two sets are centered vertically,
but align horizontally at the right
```

```
    /// edge of each set's left-most
    rectangle.] (HorizontalAlignment-3-iOS)
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct HorizontalAlignment
: Equatable {

    /// Creates a custom horizontal
    alignment of the specified type.
    ///
    /// Use this initializer to create a
    custom horizontal alignment. Define
    /// an ``AlignmentID`` type, and then
    use that type to create a new
    /// static property on
    ``HorizontalAlignment``:
    ///
    ///     private struct
OneQuarterAlignment: AlignmentID {
    ///         static func
defaultValue(in context: ViewDimensions)
-> CGFloat {
        ///             context.width / 4
        ///         }
        ///     }
        ///

        /// extension HorizontalAlignment
{
    ///         static let oneQuarter =
HorizontalAlignment(OneQuarterAlignment.s
elf)
        ///     }
        ///
```

```
    /// Every horizontal alignment
    instance that you create needs a unique
    /// identifier. For more information,
    see ``AlignmentID``.
    ///
    /// - Parameter id: The type of an
    identifier that uniquely identifies a
    /// horizontal alignment.
    public init(_ id: any
    AlignmentID.Type)

    /// You don't use this property
    directly.
    public let key: AlignmentKey

    /// 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:
    HorizontalAlignment, b:
    HorizontalAlignment) -> Bool
}
```

```
@available(iOS 16.0, macOS 13.0, tvOS
```

```
16.0, watchOS 9.0, *)
extension HorizontalAlignment {

    /// Merges a sequence of explicit
    alignment values produced by
    /// this instance.
    ///
    /// For built-in horizontal alignment
    types, this method returns the mean
    /// of all non-'nil' values.
    public func combineExplicit<S>(_
values: S) -> CGFloat? where S :
Sequence, S.Element == CGFloat?
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension HorizontalAlignment {

    /// A guide that marks the leading
    edge of the view.
    ///
    /// Use this guide to align the
    leading edges of views.
    /// For a device that uses a left-to-
    right language, the leading edge
    /// is on the left:
    ///
    /// ! [A box that contains the word,
    Leading. Vertical
    /// lines appear above and below the
    box. The lines align horizontally
    /// with the left edge of the box.]
```

```
(HorizontalAlignment-leading-1-iOS)
    /**
     /// The following code generates the
image above using a ``VStack``:
    /**
     /// struct
HorizontalAlignmentLeading: View {
    /**
         var body: some View {
    /**
VStack(alignment: .leading, spacing: 0) {
    /**
Color.red.frame(width: 1)
    /**
Text("Leading").font(.title).border(.gray
)
    /**
Color.red.frame(width: 1)
    /**
                     }
    /**
                     }
    /**
                     }
    /**
public static let leading:
HorizontalAlignment
    /**
     /// A guide that marks the horizontal
center of the view.
    /**
     /// Use this guide to align the
centers of views:
    /**
     /// ![A box that contains the word,
Center. Vertical
    /**
     /// lines appear above and below the
```

```
box. The lines align horizontally
    /// with the center of the box.]  
(HorizontalAlignmentCenter-1-iOS)
    ///
    /// The following code generates the
image above using a ``VStack``:  

    ///
    /// struct
HorizontalAlignmentCenter: View {
    ///         var body: some View {
    ///
VStack(alignment: .center, spacing: 0) {
    ///
Color.red.frame(width: 1)
    ///
Text("Center").font(.title).border(.gray)
    ///
Color.red.frame(width: 1)
    ///
        }
    ///
        }
    ///
}
///
public static let center:
HorizontalAlignment
    ///
    /// A guide that marks the trailing
edge of the view.
    ///
    /// Use this guide to align the
trailing edges of views.
    /// For a device that uses a left-to-
right language, the trailing edge
    /// is on the right:
```

```
///  
/// ! [A box that contains the word,  
Trailing. Vertical  
/// lines appear above and below the  
box. The lines align horizontally  
/// with the right edge of the box.]  
(HorizontalAlignment-trailing-1-iOS)  
///  
/// The following code generates the  
image above using a ``VStack``:  
///  
/// struct  
HorizontalAlignmentTrailing: View {  
    ///         var body: some View {  
    ///  
VStack(alignment: .trailing, spacing: 0)  
{  
    ///  
Color.red.frame(width: 1)  
    ///  
Text("Trailing").font(.title).border(.gra  
y)  
    ///  
Color.red.frame(width: 1)  
    ///         }  
    ///         }  
    ///         }  
    ///  
    public static let trailing:  
HorizontalAlignment  
}  
  
@available(iOS 16.0, macOS 13.0, *)
```

```
@available(tvOS, unavailable)
@available(watchOS, unavailable)
extension HorizontalAlignment {

    /// A guide marking the leading edge
    /// of a `List` row separator.
    ///
    /// Use this guide to align the
    /// leading end of the bottom `List` row
    /// separator with any other
    /// horizontal guide of a view that is part
    /// of the
    /// cell content.
    ///
    /// The following example shows the
    /// row separator aligned with the leading
    /// edge of the `Text` containing the
    /// name of food:
    ///
    ///     List {
    ///         ForEach(favoriteFoods)
    ///         { food in
    ///             HStack {
    ///                 Text(food.emoji)
    ///                     .font(.system
    ///                         .size: 40))
    ///                 Text(food.name)
    ///                     .alignmentGu
    /// de(.listRowSeparatorLeading) {
    ///     $0[.leading]
    ///     ///
    ///     ///
    ///     }
    ///     }
    /// }
```

```
    /**
     */
    /**
     * To change the visibility or tint
     * of the row separator use respectively
     */
``View/listRowSeparator(_:edges:)`` and
    /**
``View/listRowSeparatorTint(_:edges:)``.
    /**
public static let
listRowSeparatorLeading:
HorizontalAlignment
```

```
    /**
     * A guide marking the trailing edge
     * of a `List` row separator.
    /**
     * Use this guide to align the
     * trailing end of the bottom `List` row
     * separator with any other
     * horizontal guide of a view that is part
     * of the
     * cell content.
    /**
     * To change the visibility or tint
     * of the row separator use respectively
    /**
``View/listRowSeparator(_:edges:)`` and
    /**
``View/listRowSeparatorTint(_:edges:)``.
    /**
public static let
listRowSeparatorTrailing:
```

```
HorizontalAlignment
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension HorizontalAlignment : Sendable
{

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension HorizontalAlignment : BitwiseCopyable {
}

/// A direction on the horizontal axis.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
@frozen public enum HorizontalDirection : Int8, CaseIterable, Codable {

    /// The leading direction.
    case leading

    /// The trailing direction.
    case trailing

    /// An efficient set of horizontal
    directions.
    @frozen public struct Set : OptionSet, Equatable, Hashable {
        /// The element type of the
        
```

```
option set.  
    ///  
    /// To inherit all the default  
implementations from the `OptionSet`  
protocol,  
    /// the `Element` type must be  
`Self`, the default.  
    public typealias Element =  
HorizontalDirection.Set  
  
    /// The corresponding value of  
the raw type.  
    ///  
    /// A new instance initialized  
with `rawValue` will be equivalent to  
this  
    /// instance. For example:  
    ///  
    ///     enum PaperSize: String {  
    ///         case A4, A5, Letter,  
Legal  
    ///     }  
    ///  
    ///     let selectedSize =  
PaperSize.Letter  
    ///  
print(selectedSize.rawValue)  
    ///     // Prints "Letter"  
    ///  
    ///     print(selectedSize ==  
PaperSize(rawValue:  
selectedSize.rawValue)!)  
    ///     // Prints "true"
```

```
public let rawValue: Int8

    /// Creates a new option set from
the given raw value.
    /**
     /// This initializer always
succeeds, even if the value passed as
`rawValue`
        /// exceeds the static properties
declared as part of the option set. This
        /// example creates an instance
of `ShippingOptions` with a raw value
beyond
        /// the highest element, with a
bit mask that effectively contains all
the
        /// declared static members.
        /**
         ///     let extraOptions =
ShippingOptions(rawValue: 255)
        /**
print(extraOptions.isStrictSuperset(of: .
all))
        ///     // Prints "true"
        /**
         /// - Parameter rawValue: The raw
value of the option set to create. Each
bit
        ///     of `rawValue` potentially
represents an element of the option set,
        ///     though raw values may
include bits that are not defined as
distinct
```

```
    /// values of the `OptionSet`  
    type.  
    public init(rawValue: Int8)  
  
        /// A set containing only the  
        leading horizontal direction.  
        public static let leading:  
HorizontalDirection.Set  
  
        /// A set containing only the  
        trailing horizontal direction.  
        public static let trailing:  
HorizontalDirection.Set  
  
        /// A set containing the leading  
        and trailing horizontal directions.  
        public static let all:  
HorizontalDirection.Set  
  
        /// Creates a set of directions  
        containing only the specified direction.  
        public init(_ direction:  
HorizontalDirection)  
  
        /// The type of the elements of  
        an array literal.  
        @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
        public typealias  
ArrayLiteralElement =  
HorizontalDirection.Set.Element
```

```
    /// The raw type that can be used
    to represent all values of the conforming
    /// type.
    ///
    /// Every distinct value of the
    conforming type has a corresponding
    unique
    /// value of the `RawValue` type,
    but there may be values of the `RawValue`
    /// type that don't have a
    corresponding value of the conforming
    type.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias RawValue = Int8
}

/// Creates a new instance with the
specified raw value.
///
/// If there is no value of the type
that corresponds with the specified raw
    /// value, this initializer returns
`nil`. For example:
///
///     enum PaperSize: String {
///         case A4, A5, Letter,
Legal
    ///
///     }
    ///
///     print(PaperSize(rawValue:
"Legal"))
```

```
    ///      // Prints
"Optional("PaperSize.Legal")"
    ///
    ///      print(PaperSize(rawValue:
"Tabloid"))
    ///      // Prints "nil"
    ///
    /// - Parameter rawValue: The raw
value to use for the new instance.
public init?(rawValue: Int8)

    /// A type that can represent a
collection of all values of this type.
@available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
public typealias AllCases =
[HorizontalDirection]

    /// The raw type that can be used to
represent all values of the conforming
/// type.
///
    /// Every distinct value of the
conforming type has a corresponding
unique
    /// value of the `RawValue` type, but
there may be values of the `RawValue`
    /// type that don't have a
corresponding value of the conforming
type.
@available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
```

```
*)  
    public typealias RawValue = Int8  
  
        /// A collection of all values of  
        this type.  
        nonisolated public static var  
allCases: [HorizontalDirection] { get }  
  
        /// The corresponding value of the  
        raw type.  
        ///  
        /// A new instance initialized with  
`rawValue` will be equivalent to this  
        /// instance. For example:  
        ///  
        ///     enum PaperSize: String {  
        ///         case A4, A5, Letter,  
Legal  
        ///     }  
        ///  
        ///     let selectedSize =  
PaperSize.Letter  
        ///     print(selectedSize.rawValue)  
        ///     // Prints "Letter"  
        ///  
        ///     print(selectedSize ==  
PaperSize(rawValue:  
selectedSize.rawValue)!)  
        ///     // Prints "true"  
    public var rawValue: Int8 { get }  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS
```

```
18.0, watchOS 11.0, visionOS 2.0, *)
extension HorizontalDirection : Equatable
{
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension HorizontalDirection : Hashable
{
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension HorizontalDirection :
RawRepresentable {
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension HorizontalDirection : Sendable
{
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension HorizontalDirection :
BitwiseCopyable {
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension HorizontalDirection.Set :
Sendable {
```

```
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension HorizontalDirection.Set :
BitwiseCopyable {
}

/// An edge on the horizontal axis.
///
/// Use a horizontal edge for tasks like
/// setting a swipe action with the
///
``View/swipeActions(edge:allowsFullSwipe:
content)``
/// view modifier. The positions of the
/// leading and trailing edges
/// depend on the locale chosen by the
/// user.
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
@frozen public enum HorizontalEdge : Int8, CaseIterable, Codable {

    /// The leading edge.
    case leading

    /// The trailing edge.
    case trailing

    /// An efficient set of horizontal
    /// edges.
    @frozen public struct Set : OptionSet
```

```
{  
    /// The element type of the  
    option set.  
    ///  
    /// To inherit all the default  
    implementations from the `OptionSet`  
    protocol,  
    /// the `Element` type must be  
    `Self`, the default.  
    public typealias Element =  
HorizontalEdge.Set  
  
    /// The corresponding value of  
the raw type.  
    ///  
    /// A new instance initialized  
with `rawValue` will be equivalent to  
this  
    /// instance. For example:  
    ///  
    ///     enum PaperSize: String {  
    ///         case A4, A5, Letter,  
Legal  
    ///     }  
    ///  
    ///     let selectedSize =  
PaperSize.Letter  
    ///  
print(selectedSize.rawValue)  
    ///     // Prints "Letter"  
    ///  
    ///     print(selectedSize ==
```

```
PaperSize(rawValue:  
selectedSize.rawValue)!)  
    ///      // Prints "true"  
    public let rawValue: Int8  
  
        /// Creates a new option set from  
the given raw value.  
        ///  
        /// This initializer always  
succeeds, even if the value passed as  
'rawValue'  
        /// exceeds the static properties  
declared as part of the option set. This  
        /// example creates an instance  
of `ShippingOptions` with a raw value  
beyond  
        /// the highest element, with a  
bit mask that effectively contains all  
the  
        /// declared static members.  
        ///  
        ///     let extraOptions =  
ShippingOptions(rawValue: 255)  
        ///  
print(extraOptions.isStrictSuperset(of: .  
all))  
        ///      // Prints "true"  
        ///  
        /// - Parameter rawValue: The raw  
value of the option set to create. Each  
bit  
        ///     of `rawValue` potentially  
represents an element of the option set,
```

```
    /// though raw values may
    include bits that are not defined as
    distinct
    /// values of the `OptionSet`-
type.
public init(rawValue: Int8)

    /// A set containing only the
    leading horizontal edge.
public static let leading:
HorizontalEdge.Set

    /// A set containing only the
    trailing horizontal edge.
public static let trailing:
HorizontalEdge.Set

    /// A set containing the leading
    and trailing horizontal edges.
public static let all:
HorizontalEdge.Set

    /// Creates a set of edges
    containing only the specified horizontal
    edge.
public init(_ edge:
HorizontalEdge)

    /// The type of the elements of
    an array literal.
@available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
public typealias
```

```
ArrayLiteralElement =
HorizontalEdge.Set.Element
```

```
    /// The raw type that can be used
    to represent all values of the conforming
    /// type.
    ///
    /// Every distinct value of the
    conforming type has a corresponding
    unique
    /// value of the `RawValue` type,
    but there may be values of the `RawValue`
    /// type that don't have a
    corresponding value of the conforming
    type.
    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias RawValue = Int8
}

    /// Creates a new instance with the
    specified raw value.
    ///
    /// If there is no value of the type
    that corresponds with the specified raw
    /// value, this initializer returns
    `nil`. For example:
    ///
    ///     enum PaperSize: String {
    ///         case A4, A5, Letter,
Legal
    ///     }
    ///
```

```
    ///      print(PaperSize(rawValue:  
"Legal"))  
    ///      // Prints  
"Optional("PaperSize.Legal")"  
    ///  
    ///      print(PaperSize(rawValue:  
"Tabloid"))  
    ///      // Prints "nil"  
    ///  
    /// - Parameter rawValue: The raw  
value to use for the new instance.  
public init?(rawValue: Int8)  
  
    /// A type that can represent a  
collection of all values of this type.  
    @available(iOS 15.0, tvOS 15.0,  
watchOS 8.0, macOS 12.0, *)  
    public typealias AllCases =  
[HorizontalEdge]  
  
    /// The raw type that can be used to  
represent all values of the conforming  
    /// type.  
    ///  
    /// Every distinct value of the  
conforming type has a corresponding  
unique  
    /// value of the `RawValue` type, but  
there may be values of the `RawValue`  
    /// type that don't have a  
corresponding value of the conforming  
type.  
    @available(iOS 15.0, tvOS 15.0,
```

```
watchOS 8.0, macOS 12.0, *)
public typealias RawValue = Int8

    /// A collection of all values of
this type.
nonisolated public static var
allCases: [HorizontalEdge] { get }

    /// The corresponding value of the
raw type.
///
/// A new instance initialized with
`rawValue` will be equivalent to this
/// instance. For example:
///
///     enum PaperSize: String {
///         case A4, A5, Letter,
Legal
///
}
///
///
///     let selectedSize =
PaperSize.Letter
///
print(selectedSize.rawValue)
// Prints "Letter"
///
print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
///
// Prints "true"
public var rawValue: Int8 { get }
}

@available(iOS 15.0, macOS 12.0, tvOS
```

```
15.0, watchOS 8.0, *)
extension HorizontalEdge : Equatable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension HorizontalEdge : Hashable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension HorizontalEdge :
RawRepresentable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension HorizontalEdge : Sendable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension HorizontalEdge :
BitwiseCopyable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension HorizontalEdge.Set : Sendable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
```

```
extension HorizontalEdge.Set :  
BitwiseCopyable {  
}  
  
/// A transition that returns the input  
view, unmodified, as the output  
/// view.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
@MainActor @preconcurrency public struct  
IdentityTransition : Transition {  
  
    @MainActor @preconcurrency public  
init()  
  
    /// Gets the current body of the  
caller.  
    ///  
    /// `content` is a proxy for the view  
that will have the modifier  
    /// represented by `Self` applied to  
it.  
    @MainActor @preconcurrency public  
func body(content:  
IdentityTransition.Content, phase:  
TransitionPhase) ->  
IdentityTransition.Content  
  
    /// Returns the properties this  
transition type has.  
    ///  
    /// Defaults to  
`TransitionProperties()`.
```

```
    @MainActor @preconcurrency public
static let properties: TransitionProperties

    /// The type of view representing the
body.
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Body =
IdentityTransition.Content
}

/// A view that displays an image.
///
/// Use an `Image` instance when you want
to add images to your SwiftUI app.
/// You can create images from many
sources:
///
/// * Image files in your app's asset
library or bundle. Supported types
include
/// PNG, JPEG, HEIC, and more.
/// Instances of platform-specific
image types, like
///
<doc://com.apple.documentation/documentation/UIKit/UIImage> and
///
<doc://com.apple.documentation/documentation/AppKit/NSImage>.
/// * A bitmap stored in a Core Graphics
///
```

```
<doc://com.apple.documentation/documentation/coregraphics/cgimage>
/// instance.
/// * System graphics from the SF Symbols
set.
///
/// The following example shows how to
load an image from the app's asset
/// library or bundle and scale it to fit
within its container:
///
///     Image("Landscape_4")
///         .resizable()
///         .aspectRatio(contentMode: .fi
t)
///     Text("Water wheel")
///
/// ! [An image of a water wheel and its
adjoining building, resized to fit the
/// width of an iPhone display. The words
Water wheel appear under this
/// image.] (Image-1.png)
///
/// You can use methods on the `Image`
type as well as
/// standard view modifiers to adjust the
size of the image to fit your app's
/// interface. Here, the `Image` type's
///
``Image/resizable(capInsets:resizingMode:
)`` method scales the image to fit
/// the current view. Then, the
/// ``View/aspectRatio(_:contentMode:)``
```

```
view modifier adjusts
/// this resizing behavior to maintain
the image's original aspect ratio, rather
/// than scaling the x- and y-axes
independently to fill all four sides of
the
/// view. The article
/// <doc:Fitting-Images-into-Available-
Space> shows how to apply scaling,
/// clipping, and tiling to `Image`
instances of different sizes.
///
/// An `Image` is a late-binding token;
the system resolves its actual value
/// only when it's about to use the image
in an environment.
///
/// ### Making images accessible
///
/// To use an image as a control, use one
of the initializers that takes a
/// `label` parameter. This allows the
system's accessibility frameworks to use
/// the label as the name of the control
for users who use features like
/// VoiceOver. For images that are only
present for aesthetic reasons, use an
/// initializer with the `decorative`
parameter; the accessibility systems
/// ignore these images.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct Image : Equatable,
```

```
Sendable {

    /// 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 == (lhs: Image,
    rhs: Image) -> Bool
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Image {

    /// The modes that SwiftUI uses to
    resize an image to fit within
    /// its containing view.
    public enum ResizingMode : Sendable {

        /// A mode to repeat the image at
        its original size, as many
        /// times as necessary to fill
        the available space.
        case tile
```

```
    /// A mode to enlarge or reduce
    the size of an image so that it
    /// fills the available space.
    case stretch

        /// 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:
Image.ResizingMode, b:
Image.ResizingMode) -> Bool

        /// 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)

    /// 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 }
}
```

```
    /// Sets the mode by which SwiftUI
resizes an image to fit its space.
```

```
    /// - Parameters:
```

```
        /// - capInsets: Inset values that
indicate a portion of the image that
```

```
        /// SwiftUI doesn't resize.
```

```
        /// - resizingMode: The mode by
which SwiftUI resizes the image.
```

```
    /// - Returns: An image, with the new
resizing behavior set.
```

```
    public func resizable(capInsets:
EdgeInsets = EdgeInsets(), resizingMode:
Image.ResizingMode = .stretch) -> Image
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Image {
```

```
    /// The orientation of an image.
```

```
    ///

```

```
    /// Many image formats such as JPEG
include orientation metadata in the
```

```
    /// image data. In other cases, you
can specify image orientation
```

```
    /// in code. Properly specifying
orientation is often important both for
```

```
    /// displaying the image and for
    certain kinds of image processing.
    ///
    /// In SwiftUI, you provide an
    orientation value when initializing an
    /// ``Image`` from an existing
    ///
<doc://com.apple.documentation/documentation/coregraphics/cgimage>.
@frozen public enum Orientation :  
UInt8, CaseIterable, Hashable {  
  
    /// A value that indicates the
    original pixel data matches the image's
    /// intended display orientation.
    case up  
  
    /// A value that indicates a
    horizontal flip of the image from the
    /// orientation of its original
    pixel data.
    case upMirrored  
  
    /// A value that indicates a 180°
    rotation of the image from the
    /// orientation of its original
    pixel data.
    case down  
  
    /// A value that indicates a
    vertical flip of the image from the
    /// orientation of its original
    pixel data.
```

```
case downMirrored

    /// A value that indicates a 90°
    counterclockwise rotation from the
        /// orientation of its original
    pixel data.

case left

    /// A value that indicates a 90°
    clockwise rotation and horizontal
        /// flip of the image from the
    orientation of its original pixel
        /// data.

case leftMirrored

    /// A value that indicates a 90°
    clockwise rotation of the image from
        /// the orientation of its
    original pixel data.

case right

    /// A value that indicates a 90°
    counterclockwise rotation and
        /// horizontal flip from the
    orientation of its original pixel data.

case rightMirrored

    /// Creates a new instance with
the specified raw value.

    ///
    /// If there is no value of the
type that corresponds with the specified
raw
```

```
        /// value, this initializer
returns `nil`. For example:
    /**
     * enum PaperSize: String {
     *     case A4, A5, Letter,
Legal
    /**
    /**
     * print(PaperSize(rawValue:
"Legal"))
    /**
     * // Prints
"Optional("PaperSize.Legal")"
    /**
    /**
     * print(PaperSize(rawValue:
"Tabloid"))
    /**
     * // Prints "nil"
    /**
     * - Parameter rawValue: The raw
value to use for the new instance.
public init?(rawValue: UInt8)

        /// A type that can represent a
collection of all values of this type.
@available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
public typealias AllCases =
[Image.Orientation]

        /// The raw type that can be used
to represent all values of the conforming
        /// type.
    /**
    /**
     * Every distinct value of the
```

conforming type has a corresponding unique

```
    /// value of the `RawValue` type,  
but there may be values of the `RawValue`  
    /// type that don't have a  
corresponding value of the conforming  
type.
```

```
@available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
public typealias RawValue = UInt8
```

/// A collection of all values of  
this type.

```
nonisolated public static var  
allCases: [Image.Orientation] { get }
```

/// The corresponding value of  
the raw type.

```
///
```

/// A new instance initialized  
with `rawValue` will be equivalent to  
this

/// instance. For example:

```
///
```

```
///     enum PaperSize: String {  
///         case A4, A5, Letter,  
Legal
```

```
/// }
```

```
///
```

```
///     let selectedSize =  
PaperSize.Letter
```

```
///
```

```
print(selectedSize.rawValue)
```

```
        ///      // Prints "Letter"
        ///
        ///      print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
        ///      // Prints "true"
    public var rawValue: UInt8 {
get }
}
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Image {

    /// The level of quality for
    rendering an image that requires
    interpolation,
    /// such as a scaled image.
    ///
    /// The ``Image/interpolation(_:)`` modifier specifies the interpolation
    /// behavior when using the
    ``Image/resizable(capInsets:resizingMode:
)``
    /// modifier on an ``Image``. Use
    this behavior to prioritize rendering
    /// performance or image quality.
    public enum Interpolation : Sendable
{

    /// A value that indicates
SwiftUI doesn't interpolate image data.
```

```
case none
```

```
    /// A value that indicates a low  
    level of interpolation quality, which may  
    /// speed up image rendering.
```

```
case low
```

```
    /// A value that indicates a  
    medium level of interpolation quality,  
    /// between the low- and high-  
    quality values.
```

```
case medium
```

```
    /// A value that indicates a high  
    level of interpolation quality, which  
    /// may slow down image  
    rendering.
```

```
case high
```

```
    /// 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:  
Image.Interpolation, b:
```

```
Image.Interpolation) -> Bool
```

```
    /// 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)
```

```
    /// 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 }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Image {  
  
    /// Specifies the current level of  
quality for rendering an  
    /// image that requires  
interpolation.  
    ///  
    /// See the article <doc:Fitting-  
Images-into-Available-Space> for examples  
    /// of using `interpolation(_:)` when
```

scaling an ``Image``.

    /// – Parameter interpolation: The quality level, expressed as a value of  
    /// the `Interpolation` type, that SwiftUI applies when interpolating  
    /// an image.

    /// – Returns: An image with the given interpolation value set.

```
public func interpolation(_  
interpolation: Image.Interpolation) ->  
Image
```

    /// Specifies whether SwiftUI applies antialiasing when rendering

    /// the image.

    /// – Parameter isAntialiased: A Boolean value that specifies whether to  
    /// allow antialiasing. Pass `true` to allow antialiasing, `false` otherwise.

    /// – Returns: An image with the antialiasing behavior set.

```
public func antialiased(_  
isAntialiased: Bool) -> Image  
}
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Image {
```

    /// Sets the rendering mode for symbol images within this view.

    ///

    /// – Parameter mode: The symbol

rendering mode to use.

```
    /**
     * - Returns: A view that uses the
     * rendering mode you supply.
     */
    public func symbolRenderingMode(_
mode: SymbolRenderingMode?) -> Image
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Image {

    /**
     * Creates a labeled image based on
     * a Core Graphics image instance, usable
     * as content for controls.
     */
    /**
     * - Parameters:
     *   - cgImage: The base graphical
     * image.
     *   - scale: The scale factor for
     * the image,
     *   - with a value like `1.0`,
     * `2.0`, or `3.0`.
     *   - orientation: The orientation
     * of the image. The default is
     *   - ``Image/Orientation/up``.
     *   - label: The label associated
     * with the image. SwiftUI uses the label
     *   - for accessibility.
     */
    public init(_ cgImage: CGImage,
scale: CGFloat, orientation:
Image.Orientation = .up, label: Text)
```

```
    /// Creates an unlabeled, decorative
    image based on a Core Graphics image
    /// instance.
    ///
    /// SwiftUI ignores this image for
    accessibility purposes.
    ///
    /// - Parameters:
    ///   - cgImage: The base graphical
    image.
    ///   - scale: The scale factor for
    the image,
    ///   with a value like `1.0`,
    `2.0`, or `3.0`.
    ///   - orientation: The orientation
    of the image. The default is
    ///   ``Image/Orientation/up``.
    public init(decorative cgImage:
    CGImage, scale: CGFloat, orientation:
    Image.Orientation = .up)
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Image {

    /// Indicates whether SwiftUI renders
    an image as-is, or
    /// by using a different mode.
    ///
    /// The ``TemplateRenderingMode```
    enumeration has two cases:
    ///
```

```
``TemplateRenderingMode/original`` and
``TemplateRenderingMode/template``.

    /// The original mode renders pixels
    as they appear in the original source
    /// image. Template mode renders all
    nontransparent pixels as the
    /// foreground color, which you can
    use for purposes like creating image
    /// masks.

    ///
    /// The following example shows both
    rendering modes, as applied to an icon
    /// image of a green circle with
    darker green border:
    ///
    ///     Image("dot_green")
    ///         .renderingMode(.original)
    ///     Image("dot_green")
    ///         .renderingMode(.template)
    ///
    /// ! [Two identically-sized circle
    images. The circle on top is green
    /// with a darker green border. The
    circle at the bottom is a solid color,
    /// either white on a black
    background, or black on a white
    background,
    /// depending on the system's current
    dark mode
    /// setting.] (SwiftUI-Image-
    TemplateRenderingMode-dots.png)
    ///
    /// You also use `renderingMode` to
```

```
produce multicolored system graphics
    /// from the SF Symbols set. Use the
``TemplateRenderingMode/original``
    /// mode to apply a foreground color
to all parts of the symbol except
    /// those that have a distinct color
in the graphic. The following
    /// example shows three uses of the
`person.crop.circle.badge.plus` symbol
    /// to achieve different effects:
    ///
    /// * A default appearance with no
foreground color or template rendering
    /// mode specified. The symbol
appears all black in light mode, and all
    /// white in Dark Mode.
    /// * The multicolor behavior
achieved by using `original` template
    /// rendering mode, along with a blue
foreground color. This mode causes the
    /// graphic to override the
foreground color for distinctive parts of
the
    /// image, in this case the plus
icon.
    /// * A single-color template
behavior achieved by using `template`
    /// rendering mode with a blue
foreground color. This mode applies the
    /// foreground color to the entire
image, regardless of the user's
Appearance preferences.
    ///
```

```
///``swift
///HStack {
///    Image(systemName:
"person.crop.circle.badge.plus")
///    Image(systemName:
"person.crop.circle.badge.plus")
///        .renderingMode(.original)
///        .foregroundColor(.blue)
///    Image(systemName:
"person.crop.circle.badge.plus")
///        .renderingMode(.template)
///        .foregroundColor(.blue)
///}
///.font(.largeTitle)
///
///
/// ! [A horizontal layout of three
versions of the same symbol: a person
/// icon in a circle with a plus icon
overlaid at the bottom left. Each
/// applies a diffent set of colors
based on its rendering mode, as
/// described in the preceding
/// list.] (SwiftUI-Image-
TemplateRenderingMode-sfsymbols.png)
///
/// Use the SF Symbols app to find
system images that offer the multicolor
/// feature. Keep in mind that some
multicolor symbols use both the
/// foreground and accent colors.
///
/// - Parameter renderingMode: The
```

```
mode SwiftUI uses to render images.  
    /// - Returns: A modified ``Image``.  
    public func renderingMode(_  
renderingMode:  
Image.TemplateRenderingMode?) -> Image  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Image {  
  
    /// Creates a labeled image that you  
    can use as content for controls.  
    ////  
    /// - Parameters:  
    /// - name: The name of the image  
    resource to lookup, as well as the  
    /// localization key with which  
    to label the image.  
    /// - bundle: The bundle to search  
    for the image resource and localization  
    /// content. If `nil`, SwiftUI  
    uses the main `Bundle`. Defaults to  
    `nil`.  
    public init(_ name: String, bundle:  
Bundle? = nil)  
  
    /// Creates a labeled image that you  
    can use as content for controls, with  
    /// the specified label.  
    ////  
    /// - Parameters:  
    /// - name: The name of the image
```

```
resource to lookup
    /// - bundle: The bundle to search
for the image resource. If `nil`,
    /// SwiftUI uses the main
`Bundle`. Defaults to `nil`.
    /// - label: The label associated
with the image. SwiftUI uses the label
    /// for accessibility.
public init(_ name: String, bundle:
Bundle? = nil, label: Text)

    /// Creates an unlabeled, decorative
image.
    ///
    /// SwiftUI ignores this image for
accessibility purposes.
    ///
    /// - Parameters:
    /// - name: The name of the image
resource to lookup
    /// - bundle: The bundle to search
for the image resource. If `nil`,
    /// SwiftUI uses the main
`Bundle`. Defaults to `nil`.
public init(decorative name: String,
bundle: Bundle? = nil)

    /// Creates a system symbol image.
    ///
    /// This initializer creates an image
using a system-provided symbol. Use
    /// [SF Symbols]
(https://developer.apple.com/design/resou
```

```
rces/#sf-symbols)
    /// to find symbols and their
corresponding names.
    /**
     /// To create a custom symbol image
from your app's asset catalog, use
     /// ``Image/init(_:bundle:)``
instead.
    /**
     /// - Parameters:
     ///   - systemName: The name of the
system symbol image.
     ///   Use the SF Symbols app to
look up the names of system symbol
images.
    @available(macOS 11.0, *)
    public init(systemName: String)
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension Image {

    /// Creates a system symbol image
with a variable value.
    /**
     /// This initializer creates an image
using a system-provided symbol. The
     /// rendered symbol may alter its
appearance to represent the value
     /// provided in `variableValue`. Use
     /// [SF Symbols]
(https://developer.apple.com/design/resources/#sf-symbols)
```

```
rces/#sf-symbols)
```

```
    /// (version 4.0 or later) to find  
system symbols that support variable
```

```
    /// values and their corresponding  
names.
```

```
    ///
```

```
    /// The following example shows the  
effect of creating the `chart.bar.fill`
```

```
    /// symbol with different values.
```

```
    ///
```

```
    /// HStack{
```

```
    ///     Image(systemName:  
"chart.bar.fill", variableValue: 0.3)
```

```
    ///     Image(systemName:  
"chart.bar.fill", variableValue: 0.6)
```

```
    ///     Image(systemName:  
"chart.bar.fill", variableValue: 1.0)
```

```
    /// }
```

```
    /// .font(.system(.largeTitle))
```

```
    ///
```

```
    /// ! [Three instances of the bar  
chart symbol, arranged horizontally.
```

```
    /// The first fills one bar, the  
second fills two bars, and the last
```

```
    /// symbol fills all three bars.]
```

```
(Image-3)
```

```
    ///
```

```
    /// To create a custom symbol image  
from your app's asset
```

```
    /// catalog, use
```

```
``Image/init(_:variableValue:bundle:)``  
instead.
```

```
    ///
```

```
    /// - Parameters:  
    /// - systemName: The name of the  
    // system symbol image.  
    ///     Use the SF Symbols app to  
    look up the names of system  
    ///     symbol images.  
    /// - variableValue: An optional  
    value between `0.0` and `1.0` that  
    ///     the rendered image can use to  
    customize its appearance, if  
    ///     specified. If the symbol  
    doesn't support variable values, this  
    ///     parameter has no effect. Use  
    the SF Symbols app to look up which  
    ///     symbols support variable  
    values.  
    public init(systemName: String,  
    variableValue: Double?)  
  
    /// Creates a labeled image that you  
    can use as content for controls,  
    /// with a variable value.  
    ///  
    /// This initializer creates an image  
    using a using a symbol in the  
    /// specified bundle. The rendered  
    symbol may alter its appearance to  
    /// represent the value provided in  
    `variableValue`.  
    ///  
    /// > Note: See WWDC22 session  
    [10158: Adopt variable color in SF  
    /// Symbols]
```

```
(https://developer.apple.com/wwdc22/10158
/) for details
    /// on how to create symbols that
support variable values.
    ///
    /// - Parameters:
    ///   - name: The name of the image
resource to lookup, as well as
    ///   the localization key with
which to label the image.
    ///   - variableValue: An optional
value between `0.0` and `1.0` that
    ///   the rendered image can use to
customize its appearance, if
    ///   specified. If the symbol
doesn't support variable values, this
    ///   parameter has no effect.
    ///   - bundle: The bundle to search
for the image resource and
    ///   localization content. If
`nil`, SwiftUI uses the main
    ///   `Bundle`. Defaults to `nil`.
    ///
public init(_ name: String,
variableValue: Double?, bundle: Bundle? =
nil)

    /// Creates a labeled image that you
can use as content for controls, with
    /// the specified label and variable
value.
    ///
    /// This initializer creates an image
```

```
using a symbol in the
    /// specified bundle. The rendered
symbol may alter its appearance to
    /// represent the value provided in
`variableValue`.
    ///
    /// > Note: See WWDC22 session
[10158: Adopt variable color in SF
    /// Symbols]
(https://developer.apple.com/wwdc22/10158
/) for details on
    /// how to create symbols that
support variable values.
    ///
    /// - Parameters:
    ///   - name: The name of the image
resource to lookup.
    ///   - variableValue: An optional
value between `0.0` and `1.0` that
    ///     the rendered image can use to
customize its appearance, if
    ///     specified. If the symbol
doesn't support variable values, this
    ///     parameter has no effect.
    ///   - bundle: The bundle to search
for the image resource. If
    ///     `nil`, SwiftUI uses the main
`Bundle`. Defaults to `nil`.
    ///   - label: The label associated
with the image. SwiftUI uses
    ///     the label for accessibility.
    ///
public init(_ name: String,
```

```
variableValue: Double?, bundle: Bundle? =  
nil, label: Text)  
  
    /// Creates an unlabeled, decorative  
image, with a variable value.  
    ///  
    /// This initializer creates an image  
using a using a symbol in the  
    /// specified bundle. The rendered  
symbol may alter its appearance to  
    /// represent the value provided in  
`variableValue`.  
    ///  
    /// > Note: See WWDC22 session  
[10158: Adopt variable color in SF  
    /// Symbols]  
(https://developer.apple.com/wwdc22/10158  
/) for details on  
    /// how to create symbols that  
support variable values.  
    ///  
    /// SwiftUI ignores this image for  
accessibility purposes.  
    ///  
    /// - Parameters:  
    ///     - name: The name of the image  
resource to lookup.  
    ///     - variableValue: An optional  
value between `0.0` and `1.0` that  
    ///         the rendered image can use to  
customize its appearance, if  
    ///         specified. If the symbol  
doesn't support variable values, this
```

```
    ///      parameter has no effect.
    /// - bundle: The bundle to search
for the image resource. If
    /// `nil`, SwiftUI uses the main
`Bundle`. Defaults to `nil`.
    ///
public init(decorative name: String,
variableValue: Double?, bundle: Bundle? =
nil)
}

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

    /// Initialize an `Image` with an
image resource.
    public init(_ resource:
ImageResource)
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, *)
@available(watchOS, unavailable)
extension Image {

    public struct DynamicRange :  
Hashable, Sendable {

        /// Restrict the image content
dynamic range to the standard range.
        public static let standard:
Image.DynamicRange
```

```
    /// Allow image content to use
    some extended range. This is
        /// appropriate for placing HDR
    content next to SDR content.
    public static let
constrainedHigh: Image.DynamicRange

    /// Allow image content to use an
unrestricted extended range.
    public static let high:
Image.DynamicRange

    /// 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:  
    Image.DynamicRange, b:  
    Image.DynamicRange) -> 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 }  
}
```

```
/// Returns a new image configured  
with the specified allowed
```

```
/// dynamic range.
```

```
///
```

```
/// The following example enables HDR  
rendering for a specific
```

```
/// image view, assuming that the  
image has an HDR (ITU-R 2100)
```

```
/// color space and the output device  
supports it:
```

```
///
```

```
///     Image("hdr-  
asset").allowedDynamicRange(.high)
```

```
///
```

```
/// - Parameter range: the requested  
dynamic range, or nil to
```

```
///     restore the default allowed  
range.
```

```
///
```

```
    /// - Returns: a new image.
    public func allowedDynamicRange(_
range: Image.DynamicRange?) -> Image
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension Image {

    /// Initializes an image of the given
size, with contents provided by a
    /// custom rendering closure.
    ///
    /// Use this initializer to create an
image by calling drawing commands on a
    /// ``GraphicsContext`` provided to
the `renderer` closure.
    ///
    /// The following example shows a
custom image created by passing a
    /// `GraphicContext` to draw an
ellipse and fill it with a gradient:
    ///
    ///     let mySize = CGSize(width:
300, height: 200)
    ///     let image = Image(size:
mySize) { context in
        ///         context.fill(
        ///             Path(
        ///                 ellipseIn:
        ///                 CGRect(origin: .zero, size: mySize)),
        ///
with: .linearGradient(
```

```
    /**
     Gradient(colors: [.yellow, .orange]),
     /**
     startPoint: .zero,
     /**
     endPoint: CGPoint(x: mySize.width,
y:mySize.height))
     /**
     )
     /**
     }
     /**
     /**
     ! [An ellipse with a gradient that
blends from yellow at the upper-
     /**
     left to orange at the bottom-
right.] (Image-2)
     /**
     /**
     - Parameters:
     /**
     - size: The size of the newly-
created image.
     /**
     - label: The label associated
with the image. SwiftUI uses the label
     /**
     for accessibility.
     /**
     - opaque: A Boolean value that
indicates whether the image is fully
     /**
     opaque. This may improve
performance when `true`. Don't render
     /**
     non-opaque pixels to an image
declared as opaque. Defaults to `false`.
     /**
     - colorMode: The working color
space and storage format of the image.
     /**
     Defaults to
``ColorRenderingMode/nonLinear``.
     /**
     - renderer: A closure to draw
the contents of the image. The closure
```

```
    ///      receives a
``GraphicsContext`` as its parameter.
public init(size: CGSize, label:
Text? = nil, opaque: Bool = false,
colorMode: ColorRenderingMode
= .nonLinear, renderer: @escaping (inout
GraphicsContext) -> Void)
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Image {

    /// A type that indicates how SwiftUI
renders images.
public enum TemplateRenderingMode : Sendable {

        /// A mode that renders all non-
transparent pixels as the foreground
        /// color.
        case template

        /// A mode that renders pixels of
bitmap images as-is.
        ///
        /// For system images created
from the SF Symbol set, multicolor
symbols
        /// respect the current
foreground and accent colors.
        case original
}
```

```
    /// 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:  
Image.TemplateRenderingMode, b:  
Image.TemplateRenderingMode) -> Bool
```

```
    /// 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)

    /// 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 scale to apply to vector images
    relative to text.
    /**
     /// Use this type with the
     ``View/imageScale(_:)`` modifier, or the
     /// ``EnvironmentValues/imageScale``
     environment key, to set the image scale.
    /**
     /// The following example shows the
     three `Scale` values as applied to
     /// a system symbol image, each set
     against a text view:
    /**
     /**
      HStack { Image(systemName:
      "swift").imageScale(.small);
      Text("Small") }
      /**
      HStack { Image(systemName:
      "swift").imageScale(.medium);
      Text("Medium") }
      /**
      HStack { Image(systemName:
      "swift").imageScale(.large);
      Text("Large") }
    /**
     /**
      ! [Vertically arranged text views
      that read Small, Medium, and
      /// Large. On the left of each view
      is a system image that uses the Swift
      symbol.
      /**
      The image next to the Small text
      is slightly smaller than the text.
      /**
      The image next to the Medium text
      matches the size of the text. The
```

```
    /// image next to the Large text is
    larger than the
    /// text.](SwiftUI-
EnvironmentAdditions-Image-scale.png)
    ///
    @available(macOS 11.0, *)
    public enum Scale : Hashable,
Sendable {

    /// A scale that produces small
images.
    case small

    /// A scale that produces medium-
sized images.
    case medium

    /// A scale that produces large
images.
    case large

    /// 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:  
Image.Scale, b: Image.Scale) -> Bool  
  
        /// 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)
```

```
    /// 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 }  
}  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Image : View {  
  
    /// The type of view representing the  
body of this view.  
    ///  
    /// When you create a custom view,  
Swift infers this type from your  
    /// implementation of the required  
``View/body-swift.property`` property.
```

```
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
public typealias Body = Never
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Image.ResizingMode : Equatable
{
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Image.ResizingMode : Hashable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Image.Orientation :
RawRepresentable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Image.Orientation : Sendable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Image.Orientation :
BitwiseCopyable {
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Image.Interpolation : Equatable  
{  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Image.Interpolation : Hashable  
{  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Image.TemplateRenderingMode :  
Equatable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Image.TemplateRenderingMode :  
Hashable {  
}  
  
/// A shape style that fills a shape by  
repeating a region of an image.  
///  
/// You can also use  
``ShapeStyle/image(_:sourceRect:scale:)``  
to construct this  
/// style.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
@frozen public struct ImagePaint :  
ShapeStyle {  
  
    /// The image to be drawn.  
    public var image: Image  
  
    /// A unit-space rectangle defining  
    how much of the source image to draw.  
    ///  
    /// The results are undefined if this  
    rectangle selects areas outside the  
    /// `[0, 1]` range in either axis.  
    public var sourceRect: CGRect  
  
    /// A scale factor applied to the  
    image while being drawn.  
    public var scale: CGFloat  
  
    /// Creates a shape-filling shape  
    style.  
    ///  
    /// - Parameters:  
    ///     - image: The image to be drawn.  
    ///     - sourceRect: A unit-space  
    rectangle defining how much of the source  
    ///     image to draw. The results  
    are undefined if `sourceRect` selects  
    ///     areas outside the `[0, 1]`  
    range in either axis.  
    ///     - scale: A scale factor applied  
    to the image during rendering.  
    public init(image: Image, sourceRect:  
CGRect = CGRect(x: 0, y: 0, width: 1,
```

```
height: 1), scale: CGFloat = 1)

    /// The type of shape style this will
    resolve to.
    ///
    /// When you create a custom shape
    style, Swift infers this type
    /// from your implementation of the
    required `resolve` function.
    @available(iOS 17.0, tvOS 17.0,
    watchOS 10.0, macOS 14.0, *)
    public typealias Resolved = Never
}

/// An object that creates images from
SwiftUI views.
///
/// Use `ImageRenderer` to export bitmap
image data from a SwiftUI view. You
/// initialize the renderer with a view,
then render images on demand,
/// either by calling the
``render(rasterizationScale:renderer:)``
method, or
/// by using the renderer's properties to
create a
///
<doc://com.apple.documentation/documentation/CoreGraphics/CGImage>,
///
<doc://com.apple.documentation/documentation/AppKit/NSImage>, or
///
```

```
<doc://com.apple.documentation/documentation/UIKit/UIImage>.  
///  
/// By drawing to a ``Canvas`` and  
/// exporting with an `ImageRenderer`,  
/// you can generate images from any  
/// programmatically-rendered content, like  
/// paths, shapes, gradients, and more.  
You can also render standard SwiftUI  
/// views like ``Text`` views, or  
containers of multiple view types.  
///  
/// The following example uses a private  
`createAwardView(forUser:date:)` method  
/// to create a game app's view of a  
trophy symbol with a user name and date.  
/// This view combines a ``Canvas`` that  
applies a shadow filter with  
/// two ``Text`` views into a ``VStack``.  
A ``Button`` allows the person to  
/// save this view. The button's action  
uses an `ImageRenderer` to rasterize a  
/// `CGImage` and then calls a private  
`uploadAchievementImage(_:)` method to  
/// encode and upload the image.  
///  
///     var body: some View {  
///         let trophyAndDate =  
createAwardView(forUser: playerName,  
///  
date: achievementDate)  
///         VStack {  
///             trophyAndDate
```

```
///             Button("Save
Achievement") {
///                     let renderer =
ImageRenderer(content: trophyAndDate)
///                     if let image =
renderer.cgImage {
///
uploadAchievementImage(image)
///                     }
///                     }
///                     }
///                     }
///                     }

///     private func
createAwardView(forUser: String, date:
Date) -> some View {
///         VStack {
///             Image(systemName:
"trophy")
///                 .resizable()
///                 .frame(width: 200,
height: 200)
///                 .frame(maxWidth: .infinity,
maxHeight: .infinity)
///                 .shadow(color: .mint,
radius: 5)
///             Text(playerName)
///                 .font(.largeTitle)
///
Text(achievementDate.formatted())
///             }
///             .multilineTextAlignment(.center)
```

```
///         .frame(width: 200, height:  
290)  
///     }  
///  
/// ! [A large trophy symbol, drawn with a  
mint-colored shadow. Below this, a  
/// user name and the date and time. At  
the bottom, a button with the title  
/// Save Achievement allows people to  
save and upload an image of this  
/// view.] (ImageRenderer-1)  
///  
/// Because `ImageRenderer` conforms to  
///  
<doc://com.apple.documentation/documentation/Combine/ObservableObject>, you  
/// can use it to produce a stream of  
images as its properties change.  
Subscribe  
/// to the renderer's  
``ImageRenderer/objectWillChange``  
publisher, then use the  
/// renderer to rasterize a new image  
each time the subscriber receives an  
/// update.  
///  
/// - Important: `ImageRenderer` output  
only includes views that SwiftUI renders,  
/// such as text, images, shapes, and  
composite views of these types. It  
/// does not render views provided by  
native platform frameworks (AppKit and  
/// UIKit) such as web views, media
```

players, and some controls. For these views,

/// `ImageRenderer` displays a placeholder image, similar to the behavior of

///

``View/drawingGroup(opaque:colorMode:)``.

///

/// ### Rendering to a PDF context

///

/// The

``render(rasterizationScale:renderer:)`` method renders the specified view to any

///

<doc://com.apple.documentation/documentation/CoreGraphics/CGContext>. That means you aren't limited to creating a rasterized `CGImage`. For example, you can generate PDF data by rendering to a PDF context. The resulting PDF maintains resolution-independence for supported members of the view hierarchy, such as text, symbol images, lines, shapes, and fills.

///

/// The following example uses the `createAwardView(forUser:date:)` method from

/// the previous example, and exports its contents as an 800-by-600 point PDF to the file URL `renderURL`. It uses the `size` parameter sent to the

```
/// rendering closure to center the
`trophyAndDate` view vertically and
/// horizontally on the page.
///
///     var body: some View {
///         let trophyAndDate =
createAwardView(forUser: playerName,
///
date: achievementDate)
///         VStack {
///             trophyAndDate
///             Button("Save
Achievement") {
///                 let renderer =
ImageRenderer(content: trophyAndDate)
///                 renderer.render
{ size, renderer in
///                     var mediaBox =
CGRect(origin: .zero,
///
size: CGSize(width: 800, height: 600))
///                     guard let
consumer = CGDataConsumer(url: renderURL
as CFURL),
///                     let
pdfContext = CGContext(consumer:
consumer,
///
mediaBox: &mediaBox, nil)
///                     else {
///                         return
}
/// }
```



```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
final public class ImageRenderer<Content>  
: ObservableObject where Content : View {  
  
    /// A publisher that informs  
    subscribers of changes to the image.  
    ///  
    /// The renderer's  
    ///  
<doc://com.apple.documentation/documentation/Combine/ObservableObject/  
ObjectWillChangePublisher>  
    /// publishes `Void` elements.  
    /// Subscribers should interpret any  
    event as indicating that the contents  
    /// of the image may have changed.  
    final public let objectWillChange:  
PassthroughSubject<Void, Never>  
  
    /// If observers of this observed  
    object should be notified when the  
    /// produced image changes.  
    @available(iOS 17.2, macOS 14.2, tvOS  
17.2, watchOS 10.2, *)  
    final public var  
isObservationEnabled: Bool  
  
    /// The root view rendered by this  
    image renderer.  
    @MainActor final public var content:  
Content
```

```
    /// The size proposed to the root
view.
    /**
     /// The default value of this
property,
``ProposedViewSize/unspecified``,
    /// produces an image that matches
the original view size. You can provide
    /// a custom ``ProposedViewSize`` to
override the view's size in one or
    /// both dimensions.
@MainActor final public var
proposedSize: ProposedViewSize

    /// The scale at which to render the
image.
    /**
     /// This value is a ratio of view
points to image pixels. This relationship
    /// means that values greater than
`1.0` create an image larger than the
    /// original content view, and less
than `1.0` creates a smaller image. The
    /// following example shows a 100 x
50 rectangle view and an image rendered
    /// from it with a `scale` of `2.0` ,
resulting in an image size of
    /// 200 x 100.
    /**
     /**
         let rectangle = Rectangle()
         .frame(width: 100,
height: 50)
         let renderer =
```

```
ImageRenderer(content: rectangle)
    ///      renderer.scale = 2.0
    ///      if let rendered =
renderer.cgImage {
    ///          print("Scaled image: \
(rendered.width) x \((rendered.height)")
    ///      }
    ///      // Prints "Scaled image: 200
x 100"
    ///
    /// The default value of this
property is `1.0`.
    @MainActor final public var scale:
CGFloat

    /// A Boolean value that indicates
whether the alpha channel of the image is
    /// fully opaque.
    ///
    /// Setting this value to `true`,
meaning the alpha channel is opaque, may
    /// improve performance. Don't render
non-opaque pixels to a renderer
    /// declared as opaque. This property
defaults to `false`.
    @MainActor final public var isOpaque:
Bool

    /// The working color space and
storage format of the image.
    @MainActor final public var
colorMode: ColorRenderingMode
```

```
    /// Creates a renderer object with a
    source content view.
    ///
    /// - Parameter view: A ``View`` to
    render.
    @MainActor public init(content view:
Content)

    /// The current contents of the view,
    rasterized as a Core Graphics image.
    ///
    /// The renderer notifies its
    `objectWillChange` publisher when
    /// the contents of the image may
    have changed.
    @MainActor final public var cgImage:
CGImage? { get }

    /// Draws the renderer's current
    contents to an arbitrary Core Graphics
    /// context.
    ///
    /// Use this method to rasterize the
    renderer's content to a
    ///
<doc://com.apple.documentation/documentation/CoreGraphics/CGContext>
    /// you provide. The `renderer`
    closure receives two parameters: the
    current
    /// size of the view, and a function
    that renders the view to your
    /// `CGContext`. Implement the
```

```
closure to provide a suitable
`CGContext`,
    /// then invoke the function to
render the content to that context.
    ///
    /// - Parameters:
    ///   - rasterizationScale: The scale
factor for converting user
    ///   interface points to pixels
when rasterizing parts of the
    ///   view that can't be
represented as native Core Graphics
drawing
    ///   commands.
    ///   - renderer: The closure that
sets up the Core Graphics context and
    ///   renders the view. This
closure receives two parameters: the size
of
    ///   the view and a function that
you invoke in the closure to render the
    ///   view at the reported size.
This function takes a
    ///
<doc://com.apple.documentation/documentation/CoreGraphics/CGContext>
    /// parameter, and assumes a
bottom-left coordinate space origin.
    @MainActor final public func
render(rasterizationScale: CGFloat = 1,
renderer: (CGSize, (CGContext) -> Void)
-> Void)
```

```
    /// The type of publisher that emits
    /// before the object has changed.
    @available(iOS 16.0, tvOS 16.0,
    watchOS 9.0, macOS 13.0, *)
    public typealias
ObjectWillChangePublisher =
PassthroughSubject<Void, Never>
}

@available(iOS 17.1, macOS 14.1, tvOS
17.1, watchOS 10.1, *)
extension ImageRenderer : Observable {
}

/// A shape type that is able to inset
/// itself to produce another shape.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public protocol InsettableShape : Shape {

    /// The type of the inset shape.
    associatedtype InsetShape :
InsettableShape

    /// Returns `self` inset by `amount`.
    nonisolated func inset(by amount:
CGFloat) -> Self.InsetShape
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension InsettableShape {
```

```
    /// Returns a view that is the result
of insetting `self` by
    /// `style.lineWidth / 2`, stroking
the resulting shape with
    /// `style`, and then filling with
`content`.
    @inlinable public func
strokeBorder<S>(_ content: S, style:
StrokeStyle, antialiased: Bool = true) ->
some View where S : ShapeStyle
```

```
    /// Returns a view that is the result
of insetting `self` by
    /// `style.lineWidth / 2`, stroking
the resulting shape with
    /// `style`, and then filling with
the foreground color.
    @inlinable public func
strokeBorder(style: StrokeStyle,
antialiased: Bool = true) -> some View
```

```
    /// Returns a view that is the result
of filling the `lineWidth`-sized
    /// border (aka inner stroke) of
`self` with `content`. This is
    /// equivalent to insetting `self` by
`lineWidth / 2` and stroking the
    /// resulting shape with `lineWidth`
as the line-width.
    @inlinable public func
strokeBorder<S>(_ content: S, lineWidth:
```

```
CGFloat = 1, antialiased: Bool = true) ->
some View where S : ShapeStyle
```

```
    /// Returns a view that is the result
    /// of filling the `lineWidth`-sized
    /// border (aka inner stroke) of
    /// `self` with the foreground color.
    /// This is equivalent to insetting
    /// `self` by `lineWidth / 2` and
    /// stroking the resulting shape with
    /// `lineWidth` as the line-width.
    @inlinable public func
strokeBorder(lineWidth: CGFloat = 1,
antialiased: Bool = true) -> some View

}
```

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

    /// Returns a view that is the result
    /// of insetting `self` by
    /// `style.lineWidth / 2`, stroking
    /// the resulting shape with
    /// `style`, and then filling with
    /// `content`.
    nonisolated public func
strokeBorder<S>(_ content: S
= .foreground, style: StrokeStyle,
antialiased: Bool = true) ->
StrokeBorderShapeView<Self, S, EmptyView>
```

where S : ShapeStyle

```
    /// Returns a view that is the result  
of filling the `lineWidth`-sized  
    /// border (aka inner stroke) of  
`self` with `content`. This is  
    /// equivalent to insetting `self` by  
`lineWidth / 2` and stroking the  
    /// resulting shape with `lineWidth`  
as the line-width.
```

```
nonisolated public func  
strokeBorder<S>(_ content: S  
= .foreground, lineWidth: CGFloat = 1,  
antialiased: Bool = true) ->  
StrokeBorderStyleView<Self, S, EmptyView>  
where S : ShapeStyle  
}
```

```
/// A container that animates its content  
with keyframes.  
///  
/// The `content` closure updates every  
frame while  
/// animating, so avoid performing any  
expensive operations directly within  
/// `content`.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
public struct KeyframeAnimator<Value,  
KeyframePath, Content> : View where Value  
== KeyframePath.Value, KeyframePath :  
Keyframes, Content : View {
```

```
    /// Plays the given keyframes when
    the given trigger value changes, updating
        /// the view using the modifiers you
    apply in `body`.
    ///
    /// Note that the `content` closure
    will be updated on every frame while
        /// animating, so avoid performing
    any expensive operations directly within
        /// `content`.
    ///
    /// If the trigger value changes
    while animating, the `keyframes` closure
        /// will be called with the current
    interpolated value, and the keyframes
        /// that you return define a new
    animation that replaces the old one. The
        /// previous velocity will be
    preserved, so cubic or spring keyframes
    will
        /// maintain continuity from the
    previous animation if they do not specify
        /// a custom initial velocity.
    ///
    /// When a keyframe animation
finishes, the animator will remain at the
        /// end value, which becomes the
    initial value for the next animation.
    ///
    /// - Parameters:
    ///     - initialValue: The initial
value that the keyframes will animate
        ///     from.
```

```
    /// - trigger: A value to observe
for changes.
    /// - content: A view builder
closure that takes the interpolated value
    /// generated by the keyframes as
its single argument.
    /// - keyframes: Keyframes defining
how the value changes over time. The
    /// current value of the animator
is the single argument, which is
    /// equal to `initialValue` when
the view first appears, then is equal
    /// to the end value of the
previous keyframe animation on subsequent
    /// calls.
public init(initialValue: Value,
trigger: some Equatable, @ViewBuilder
content: @escaping (Value) -> Content,
@KeyframesBuilder<Value> keyframes:
@escaping (Value) -> KeyframePath)

    /// Loops the given keyframes
continuously, updating
    /// the view using the modifiers you
apply in `body`.
    ///
    /// Note that the `content` closure
will be updated on every frame while
    /// animating, so avoid performing
any expensive operations directly within
    /// `content`.
    ///
    /// - Parameters:
```

```
    /// - initialValue: The initial  
    value that the keyframes will animate  
    /// from.  
    /// - repeating: Whether the  
keyframes are currently repeating. If  
false,  
        /// the value at the beginning of  
the keyframe timeline will be  
        /// provided to the content  
closure.  
        /// - content: A view builder  
closure that takes the interpolated value  
        /// generated by the keyframes as  
its single argument.  
        /// - keyframes: Keyframes defining  
how the value changes over time. The  
        /// current value of the animator  
is the single argument, which is  
        /// equal to `initialValue` when  
the view first appears, then is equal  
        /// to the end value of the  
previous keyframe animation on subsequent  
        /// calls.  
    public init(initialValue: Value,  
repeating: Bool = true, @ViewBuilder  
content: @escaping (Value) -> Content,  
@KeyframesBuilder<Value> keyframes:  
@escaping (Value) -> KeyframePath)  
  
    /// The type of view representing the  
body of this view.  
    ///  
    /// When you create a custom view,
```

```
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
@available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
public typealias Body = Never
}

/// A description of how a value changes
over time, modeled using keyframes.
///
/// Unlike other animations in SwiftUI
(using ``Animation``), keyframes
/// don't interpolate between from and to
values that SwiftUI provides as
/// state changes. Instead, keyframes
fully define the path that a value
/// takes over time using the tracks that
make up their body.
///
/// `Keyframes` values are roughly
analogous to video clips;
/// they have a set duration, and you can
scrub and evaluate them for any
/// time within the duration.
///
/// The `Keyframes` structure also allows
you to compute an interpolated
/// value at a specific time, which you
can use when integrating keyframes
/// into custom use cases.
///
/// For example, you can use a
```

```
`Keyframes` instance to define animations  
for a  
/// type conforming to `Animatable`  
///  
///     let keyframes =  
KeyframeTimeline(initialValue:  
CGPoint.zero) {  
///         CubicKeyframe(.init(x: 0, y:  
100), duration: 0.3)  
///         CubicKeyframe(.init(x: 0, y:  
0), duration: 0.7)  
///     }  
///  
///     let value = keyframes.value(time:  
0.45  
///  
/// For animations that involve multiple  
coordinated changes, you can include  
/// multiple nested tracks:  
///  
///     struct Values {  
///         var rotation = Angle.zero  
///         var scale = 1.0  
///     }  
///  
///     let keyframes =  
KeyframeTimeline(initialValue: Values())  
{  
///         KeyframeTrack(\.rotation) {  
///             CubicKeyframe(.zero,  
duration: 0.2)  
///             CubicKeyframe(.degrees(45), duration:
```

```
0.3)
    /**
     * KeyframeTrack(\.scale) {
     *     CubicKeyframe(value: 1.2,
     * duration: 0.5)
     *     CubicKeyframe(value: 0.9,
     * duration: 0.2)
     *     CubicKeyframe(value: 1.0,
     * duration: 0.3)
     * }
     */
    /**
     * Multiple nested tracks update the
     * initial value in the order that they are
     * declared. This means that if multiple
     * nested plans change the same property
     * of the root value, the value from the
     * last competing track will be used.
     */
    @available(iOS 17.0, macOS 14.0, tvOS
    17.0, watchOS 10.0, *)
    public struct KeyframeTimeline<Value> {

        /**
         * Creates a new instance using the
         * initial value and content that you
         * provide.
         */
        public init(initialValue: Value,
        @KeyframesBuilder<Value> content: () ->
        some Keyframes<Value>)

        /**
         * The duration of the content in
         * seconds.
         */
        public var duration: TimeInterval {
```

```
get }

    /// Returns the interpolated value at
    the given time.
    public func value(time: Double) ->
Value

    /// Returns the interpolated value at
    the given progress in the range zero to
    one.
    public func value(progress: Double)
-> Value
}

/// A sequence of keyframes animating a
single property of a root type.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct KeyframeTrack<Root, Value,
Content> : Keyframes where Value ==
Content.Value, Content :
KeyframeTrackContent {

    /// Creates an instance that animates
    the entire value from the root of the key
    path.
    ///
    /// - Parameter keyframes: A keyframe
    collection builder closure containing
    /// the keyframes that control the
    interpolation curve.
    public
init(@KeyframeTrackContentBuilder<Root>
```

```
content: () -> Content) where Root == Value

    /// Creates an instance that animates
    the property of the root value
    /// at the given key path.
    ///
    /// - Parameter keyPath: The property
    to animate.
    /// - Parameter keyframes: A keyframe
    collection builder closure containing
    /// the keyframes that control the
    interpolation curve.
    public init(_ keyPath:
WritableKeyPath<Root, Value>,
@KeyframeTrackContentBuilder<Value>
content: () -> Content)

    /// The type of keyframes
    representing the body of this type.
    ///
    /// When you create a custom
    keyframes type, Swift infers this type
    from your
    /// implementation of the required
    /// ``Keyframes/body-swift.property``
    property.
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Body = Never
}

/// A group of keyframes that define an
```

```
interpolation curve of an animatable
/// value.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public protocol
KeyframeTrackContent<Value> {

    associatedtype Value : Animatable =
Self.Body.Value

    associatedtype Body :
KeyframeTrackContent

    /// The composition of content that
comprise the keyframe track.

@KeyframeTrackContentBuilder<Self.Value>
var body: Self.Body { get }
}

/// The builder that creates keyframe
track content from the keyframes
/// that you define within a closure.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@resultBuilder public struct
KeyframeTrackContentBuilder<Value> where
Value : Animatable {

    public static func
buildExpression<K>(_ expression: K) -> K
where Value == K.Value, K :
KeyframeTrackContent
```

```
    public static func buildArray(_  
components: [some  
KeyframeTrackContent<Value>]) -> some  
KeyframeTrackContent<Value>
```

```
    public static func buildEither<First,  
Second>(first component: First) ->  
KeyframeTrackContentBuilder<Value>.Conditional<Value, First, Second> where Value  
== First.Value, First :  
KeyframeTrackContent, Second :  
KeyframeTrackContent, First.Value ==  
Second.Value
```

```
    public static func buildEither<First,  
Second>(second component: Second) ->  
KeyframeTrackContentBuilder<Value>.Conditional<Value, First, Second> where Value  
== First.Value, First :  
KeyframeTrackContent, Second :  
KeyframeTrackContent, First.Value ==  
Second.Value
```

```
    public static func  
buildPartialBlock<K>(first: K) -> K where  
Value == K.Value, K :  
KeyframeTrackContent
```

```
    public static func  
buildPartialBlock(accumulated: some  
KeyframeTrackContent<Value>, next: some
```

```
KeyframeTrackContent<Value>) -> some
KeyframeTrackContent<Value>

    public static func buildBlock() ->
some KeyframeTrackContent<Value>

}

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

    /// A conditional result from the
result builder.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public struct
Conditional<ConditionalValue, First,
Second> : KeyframeTrackContent where
ConditionalValue == First.Value, First :
KeyframeTrackContent, Second :
KeyframeTrackContent, First.Value ==
Second.Value {

        @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
        public typealias Body =
KeyframeTrackContentBuilder<KeyframeTrack
ContentBuilder<Value>.Conditional<Conditi
onalValue, First,
Second>.Value>.Conditional<ConditionalVal
ue, First, Second>
```

```
        @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Value =
ConditionalValue
}
}

/// A type that defines changes to a
value over time.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public protocol Keyframes<Value> {

    /// The type of value animated by
    this keyframes type
    associatedtype Value =
Self.Body.Value

    /// The type of keyframes
    representing the body of this type.
    ///
    /// When you create a custom
    keyframes type, Swift infers this type
    from your
    /// implementation of the required
    /// ``Keyframes/body-swift.property``
    property.
    associatedtype Body : Keyframes

    /// The composition of content that
    comprise the keyframes.
    @KeyframesBuilder<Self.Value> var
```

```
body: Self.Body { get }  
}  
  
/// A builder that combines keyframe  
content values into a single value.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
@resultBuilder public struct  
KeyframesBuilder<Value> {  
  
    public static func  
buildExpression<K>(_ expression: K) -> K  
where Value == K.Value, K :  
KeyframeTrackContent  
  
    public static func buildArray(_  
components: [some  
KeyframeTrackContent<Value>]) -> some  
KeyframeTrackContent<Value>  
  
    public static func buildEither<First,  
Second>(first component: First) ->  
KeyframeTrackContentBuilder<Value>.Condit  
ional<Value, First, Second> where Value  
== First.Value, First :  
KeyframeTrackContent, Second :  
KeyframeTrackContent, First.Value ==  
Second.Value  
  
    public static func buildEither<First,  
Second>(second component: Second) ->  
KeyframeTrackContentBuilder<Value>.Condit
```

```
ional<Value, First, Second> where Value
== First.Value, First :
KeyframeTrackContent, Second :
KeyframeTrackContent, First.Value ==
Second.Value

    public static func
buildPartialBlock<K>(first: K) -> K where
Value == K.Value, K :
KeyframeTrackContent

    public static func
buildPartialBlock(accumulated: some
KeyframeTrackContent<Value>, next: some
KeyframeTrackContent<Value>) -> some
KeyframeTrackContent<Value>

    public static func buildBlock() ->
some KeyframeTrackContent<Value> where
Value : Animatable

    public static func
buildFinalResult<Content>(_ component:
Content) -> KeyframeTrack<Value, Value,
Content> where Value == Content.Value,
Content : KeyframeTrackContent

    /// Keyframes
    public static func
buildExpression<Content>(_ expression:
Content) -> Content where Value ==
```

```
Content.Value, Content : Keyframes
```

```
    public static func  
buildPartialBlock<Content>(first:  
Content) -> Content where Value ==  
Content.Value, Content : Keyframes
```

```
    public static func  
buildPartialBlock(accumulated: some  
Keyframes<Value>, next: some  
Keyframes<Value>) -> some  
Keyframes<Value>
```

```
    public static func buildBlock() ->  
some Keyframes<Value>
```

```
    public static func  
buildFinalResult<Content>(_ component:  
Content) -> Content where Value ==  
Content.Value, Content : Keyframes  
}
```

```
/// A type that defines the geometry of a  
collection of views.  
///  
/// You traditionally arrange views in  
your app's user interface using built-in  
/// layout containers like ``HStack`` and  
``Grid``. If you need more complex  
/// layout behavior, you can define a  
custom layout container by creating a
```

```
type
/// that conforms to the `Layout` protocol and implementing its required
/// methods:
///
/// *
``Layout/sizeThatFits(proposal:subviews:cache:)``
/// reports the size of the composite layout view.
/// *
``Layout/placeSubviews(in:proposal:subviews:cache:)``
/// assigns positions to the container's subviews.
///
/// You can define a basic layout type with only these two methods:
///
///     struct BasicVStack: Layout {
///         func sizeThatFits(
///             proposal:
///             ProposedViewSize,
///             subviews: Subviews,
///             cache: inout ())
///         ) -> CGSize {
///             // Calculate and return the size of the layout container.
///         }
///
///         func placeSubviews(
///             in bounds: CGRect,
///             proposal:
```

```
ProposedViewSize,  
///           subviews: Subviews,  
///           cache: inout ()  
/// {  
///     // Tell each subview  
///     // where to appear.  
/// }  
/// }  
  
/// Use your layout the same way you use  
/// a built-in layout  
/// container, by providing a  
``ViewBuilder`` with the list of subviews  
/// to arrange:  
///  
///     BasicVStack {  
///         Text("A Subview")  
///         Text("Another Subview")  
///     }  
  
/// ### Support additional behaviors  
///  
/// You can optionally implement other  
/// protocol methods and properties to  
/// provide more layout container  
/// features:  
///  
/// * Define explicit horizontal and  
/// vertical layout guides for the container  
/// by  
///     implementing  
``explicitAlignment(of:in:proposal:subvie  
ws:cache:)``
```

```
/// for each dimension.  
/// * Establish the preferred spacing  
around the container by implementing  
/// ``spacing(subviews:cache:)``.  
/// * Indicate the axis of orientation  
for a container that has characteristics  
/// of a stack by implementing the  
``layoutProperties-5rb5b`` static  
property.  
/// * Create and manage a cache to store  
computed values across different  
/// layout protocol calls by  
implementing ``makeCache(subviews:)``.  
///  
/// The protocol provides default  
implementations for these symbols  
/// if you don't implement them. See each  
method or property for details.  
///  
/// ### Add input parameters  
///  
/// You can define parameters as inputs  
to the layout, like you might  
/// for a ``View``:  
///  
///     struct BasicVStack: Layout {  
///         var alignment:  
HorizontalAlignment  
///         // ...  
///     }  
///  
/// Set the parameters at the point where
```

```
you instantiate the layout:  
///  
///     BasicVStack(alignment: .leading)  
{  
///         // ...  
///     }  
///  
/// If the layout provides default values  
for its parameters, you can omit the  
/// parameters at the call site, but you  
might need to keep the parentheses  
/// after the name of the layout,  
depending on how you specify the  
defaults.  
/// For example, suppose you set a  
default alignment for the basic stack in  
/// the parameter declaration:  
///  
///     struct BasicVStack: Layout {  
///         var alignment:  
HorizontalAlignment = .center  
///  
///         // ...  
///     }  
///  
/// To instantiate this layout using the  
default center alignment, you don't  
/// have to specify the alignment value,  
but you do need to add empty  
/// parentheses:  
///  
///     BasicVStack() {  
///         // ...
```

```
///      }
///
/// The Swift compiler requires the
/// parentheses in this case because of how
/// the
/// layout protocol implements this call
/// site syntax. Specifically, the layout's
/// ``callAsFunction(_:)`` method looks
/// for an initializer with exactly zero
/// input arguments when you omit the
/// parentheses from the call site.
/// You can enable the simpler call site
/// for a layout that doesn't have an
/// implicit initializer of this type by
/// explicitly defining one:
///
///     init() {
///         self.alignment = .center
///     }
///
/// For information about Swift
/// initializers, see
/// [Initialization]
/// (https://docs.swift.org/swift-book/LanguageGuide/Initialization.html)
/// in *The Swift Programming Language*.
///
/// ### Interact with subviews through
/// their proxies
///
/// To perform layout, you need
/// information about all of its subviews,
/// which
```

```
/// are the views that your container
arranges. While your layout can't
/// interact directly with its subviews,
it can access a set of subview proxies
/// through the ``Subviews`` collection
that each protocol method receives as
/// an input parameter. That type is an
alias for the ``LayoutSubviews``
/// collection type, which in turn
contains ``LayoutSubview`` instances
/// that are the subview proxies.
///
/// You can get information about each
subview from its proxy, like its
/// dimensions and spacing preferences.
This enables
/// you to measure subviews before you
commit to placing them. You also
/// assign a position to each subview by
calling its proxy's
///
``LayoutSubview/place(at:anchor:proposal:
)`` method.
/// Call the method on each subview from
within your implementation of the
/// layout's
``placeSubviews(in:proposal:subviews:cach
e:)`` method.
///
/// ### Access layout values
///
/// Views have layout values that you set
with view modifiers.
```

```
/// Layout containers can choose to
/// condition their behavior accordingly.
/// For example, a built-in ``HStack``
/// allocates space to its subviews based
/// in part on the priorities that you
/// set with the ``View/layoutPriority(_:)``
/// view modifier. Your layout container
/// accesses this value for a subview by
/// reading the proxy's
/// ``LayoutSubview/priority`` property.
///
/// You can also create custom layout
/// values by creating a layout key.
/// Set a value on a view with the
/// ``View/layoutValue(key:value:)`` view
/// modifier. Read the corresponding
/// value from the subview's proxy using the
/// key as an index on the subview. For
/// more information about creating,
/// setting, and accessing custom layout
/// values, see ``LayoutValueKey``.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public protocol Layout : Animatable {

    /// Properties of a layout container.
    ///
    /// Implement this property in a type
    /// that conforms to the ``Layout``
    /// protocol to characterize your
    /// custom layout container. For example,
    /// you can indicate that your layout
    /// has a vertical
```

```
    /**
     ``LayoutProperties/stackOrientation``:
    /**
     /**
      extension BasicVStack {
      /**
       static var
layoutProperties: LayoutProperties {
      /**
           var properties =
LayoutProperties()
      /**
properties.stackOrientation = .vertical
      /**
           return properties
      /**
           }
      /**
           }
      /**
      /**
       /**
        If you don't implement this
property in your custom layout, the
protocol
      /**
       provides a default
implementation, namely
``layoutProperties-6h7w0``,
      /**
       that returns a
``LayoutProperties`` instance with
default values.
      static var layoutProperties:
LayoutProperties { get }

      /**
       Cached values associated with the
layout instance.
      /**
      /**
       If you create a cache for your
custom layout, you can use
      /**
       a type alias to define this type
as your data storage type.
```

```
    /// Alternatively, you can refer to
    the data storage type directly in all
    /// the places where you work with
    the cache.
    ///
    /// See ``makeCache(subviews:)`` for
more information.

associatedtype Cache = Void

    /// A collection of proxies for the
subviews of a layout view.
    ///
    /// This collection doesn't store
views. Instead it stores instances of
    /// ``LayoutSubview``, each of which
acts as a proxy for one of the
    /// views arranged by the layout. Use
the proxies to
    /// get information about the views,
and to tell the views where to
    /// appear.
    ///
    /// For more information about the
behavior of the underlying
    /// collection type, see
``LayoutSubviews``.

typealias Subviews = LayoutSubviews

    /// Creates and initializes a cache
for a layout instance.
    ///
    /// You can optionally use a cache to
preserve calculated values across
```

```
    /// calls to a layout container's
    /// methods. Many layout types don't need
    /// a cache, because SwiftUI
    /// automatically reuses both the results of
    /// calls into the layout and the
    /// values that the layout reads from its
    /// subviews. Rely on the protocol's
    /// default implementation of this method
    /// if you don't need a cache.
    ///
    /// However you might find a cache
    /// useful when:
    ///
    /// - The layout container repeats
    /// complex, intermediate calculations
    /// across calls like
``sizeThatFits(proposal:subviews:cache:)``
,
    ///
``placeSubviews(in:proposal:subviews:cache:)``, and
    ///
``explicitAlignment(of:in:proposal:subviews:cache:)``.

    /// You might be able to improve
    /// performance by calculating values
    /// once and storing them in a cache.
    /// - The layout container reads many
``LayoutValueKey`` values from
    /// subviews. It might be more
    /// efficient to do that once and store the
    /// results in the cache, rather than
    /// rereading the subviews' values before
```

```
    /// each layout call.  
    /// - You want to maintain working  
storage, like temporary Swift arrays,  
    /// across calls into the layout, to  
minimize the number of allocation  
    /// events.  
    ///  
    /// Only implement a cache if  
profiling shows that it improves  
performance.  
    ///  
    /// Initialize a cache  
    ///  
    /// Implement the  
`makeCache(subviews:)` method to create a  
cache.  
    /// You can add computed values to  
the cache right away, using information  
    /// from the `subviews` input  
parameter, or you can do that later. The  
    /// methods of the ``Layout``  
protocol that can access the cache  
    /// take the cache as an in-out  
parameter, which enables you to modify  
    /// the cache anywhere that you can  
read it.  
    ///  
    /// You can use any storage type that  
makes sense for your layout  
    /// algorithm, but be sure that you  
only store data that you derive  
    /// from the layout and its subviews  
(lazily, if possible). For this to
```

```
    /// work correctly, SwiftUI needs to
    be able to call this method to
        /// recreate the cache without
    changing the layout result.
    ///
    /// When you return a cache from this
method, you implicitly define a type
        /// for your cache. Be sure to either
make the type of the `cache`
    /// parameters on your other
``Layout`` protocol methods match, or use
        /// a type alias to define the
``Cache`` associated type.
    ///
    /// Update the cache
    ///
    /// If the layout container or any of
its subviews change, SwiftUI
        /// calls the
``updateCache(_:subviews:)`` method so
you can
    /// modify or invalidate the contents
of the
        /// cache. The default implementation
of that method calls the
    /// ``makeCache(subviews:)`` method to
recreate the cache, but you can
    /// provide your own implementation
of the update method to take an
        /// incremental approach, if
appropriate.
    ///
    /// - Parameters:
```

```
    /// - subviews: A collection of
proxy instances that represent the
    /// views that the container
arranges. You can use the proxies in the
    /// collection to get information
about the subviews as you
    /// calculate values to store in
the cache.
```

```
    ///
    /// - Returns: Storage for calculated
data that you share among
    /// the methods of your custom
layout container.
```

```
func makeCache(subviews:
Self.Subviews) -> Self.Cache
```

```
    /// Updates the layout's cache when
something changes.
```

```
    ///
    /// If your custom layout container
creates a cache by implementing the
    /// ``makeCache(subviews:)`` method,
SwiftUI calls the update method
```

```
    /// when your layout or its subviews
change, giving you an opportunity
    /// to modify or invalidate the
contents of the cache.
```

```
    /// The method's default
implementation recreates the
    /// cache by calling the
``makeCache(subviews:)`` method,
    /// but you can provide your own
implementation to take an
```

```
    /// incremental approach, if
    appropriate.
    /**
     * - Parameters:
     *   - cache: Storage for calculated
     *     data that you share among
     *     the methods of your custom
     *     layout container.
     *   - subviews: A collection of
     *     proxy instances that represent the
     *     views arranged by the
     *     container. You can use the proxies in the
     *     collection to get information
     *     about the subviews as you
     *     calculate values to store in
     *     the cache.
     */
    func updateCache(_ cache: inout
Self.Cache, subviews: Self.Subviews)

    /// Returns the preferred spacing
    values of the composite view.
    /**
     * Implement this method to provide
     * custom spacing preferences
     * for a layout container. The value
     * you return affects
     * the spacing around the container,
     * but it doesn't affect how the
     * container arranges subviews
     * relative to one another inside the
     * container.
     */
    /**
     * Create a custom ``ViewSpacing``
```

```
instance for your container by
    /// initializing one with default
values, and then merging that with
    /// spacing instances of certain
subviews. For example, if you define
    /// a basic vertical stack that
places subviews in a column, you could
    /// use the spacing preferences of
the subview edges that make
    /// contact with the container's
edges:
    ///
    /// extension BasicVStack {
    ///     func spacing(subviews:
Subviews, cache: inout ()) -> ViewSpacing
{
    ///                     var spacing =
ViewSpacing()
    ///
    ///                     for index in
subviews.indices {
    ///                         var edges:
Edge.Set = [.leading, .trailing]
    ///                         if index == 0
{ edges.formUnion(.top) }
    ///                         if index ==
subviews.count - 1
{ edges.formUnion(.bottom) }
    ///
spacing.formUnion(subviews[index].spacing
, edges: edges)
    ///
    /// }
```

```
    /**
     *      return spacing
     */
    }

}

}

/// In the above example, the first
and last subviews contribute to the
/// spacing above and below the
container, respectively, while all
subviews
    /// affect the spacing on the leading
and trailing edges.
///
/// If you don't implement this
method, the protocol provides a default
    /// implementation that merges the
spacing preferences across all subviews
on all edges.
///
/// - Parameters:
///   - subviews: A collection of
proxy instances that represent the
    /// views that the container
arranges. You can use the proxies in the
    /// collection to get information
about the subviews as you determine
    /// how much spacing the
container prefers around it.
    /// - cache: Optional storage for
calculated data that you can share among
    /// the methods of your custom
layout container. See
    /// ``makeCache(subviews:)`` for
details.
```

```
///  
/// - Returns: A ``ViewSpacing``  
instance that describes the preferred  
/// spacing around the container  
view.  
func spacing(subviews: Self.Subviews,  
cache: inout Self.Cache) -> ViewSpacing  
  
    /// Returns the size of the composite  
view, given a proposed size  
    /// and the view's subviews.  
    ///  
    /// Implement this method to tell  
your custom layout container's parent  
    /// view how much space the container  
needs for a set of subviews, given  
    /// a size proposal. The parent might  
call this method more than once  
    /// during a layout pass with  
different proposed sizes to test the  
    /// flexibility of the container,  
using proposals like:  
    ///  
    /// * The ``ProposedViewSize/zero``  
proposal; respond with the  
    /// layout's minimum size.  
    /// * The  
``ProposedViewSize/infinity`` proposal;  
respond with the  
    /// layout's maximum size.  
    /// * The  
``ProposedViewSize/unspecified``  
proposal; respond with the
```

```
    /// layout's ideal size.  
    ///  
    /// The parent might also choose to  
    test flexibility in one dimension at a  
    /// time. For example, a horizontal  
    stack might propose a fixed height and  
    /// an infinite width, and then the  
    same height with a zero width.  
    ///  
    /// The following example calculates  
    the size for a basic vertical stack  
    /// that places views in a column,  
    with no spacing between the views:  
    ///  
    /// private struct BasicVStack:  
Layout {  
    /// func sizeThatFits(  
    ///     proposal:  
ProposedViewSize,  
    ///         subviews: Subviews,  
    ///         cache: inout ())  
    ///     ) -> CGSize {  
    ///  
    subviews.reduce(CGSize.zero) { result,  
    subview in  
        /// let size =  
        subview.sizeThatFits(.unspecified)  
        /// return CGSize(  
        ///     width:  
max(result.width, size.width),  
        ///     height:  
result.height + size.height)  
        /// }
```

```
    /**
     */
    /**
     *          // This layout also needs
a placeSubviews() implementation.
    /**
     */
    /**
     *          // The implementation asks each
subview for its ideal size by calling the
    /**
``LayoutSubview/sizeThatFits(_:)`` method
with an
    /**
     * ``ProposedViewSize/unspecified`` proposed size.
    /**
     * It then reduces these values into
a single size that represents
    /**
     * the maximum subview width and the
sum of subview heights.
    /**
     * Because this example isn't
flexible, it ignores its size proposal
    /**
     * input and always returns the same
value for a given set of subviews.
    /**
``SwiftUI views choose their own
size, so the layout engine always
    /**
     * uses a value that you return from
this method as the actual size of the
    /**
     * composite view. That size factors
into the construction of the `bounds`
    /**
     * input to the
``placeSubviews(in:proposal:subviews:cach
e:)`` method.
    /**
    /**
     * - Parameters:
```

```
    /// - proposal: A size proposal for  
the container. The container's parent  
    /// view that calls this method  
might call the method more than once  
    /// with different proposals to  
learn more about the container's  
    /// flexibility before deciding  
which proposal to use for placement.  
    /// - subviews: A collection of  
proxies that represent the  
    /// views that the container  
arranges. You can use the proxies in the  
    /// collection to get information  
about the subviews as you determine  
    /// how much space the container  
needs to display them.  
    /// - cache: Optional storage for  
calculated data that you can share among  
    /// the methods of your custom  
layout container. See  
    /// ``makeCache(subviews:)`` for  
details.  
    ///  
    /// - Returns: A size that indicates  
how much space the container  
    /// needs to arrange its subviews.  
func sizeThatFits(proposal:  
ProposedViewSize, subviews:  
Self.Subviews, cache: inout Self.Cache)  
-> CGSize  
  
    /// Assigns positions to each of the  
layout's subviews.
```

```
///  
/// SwiftUI calls your implementation  
of this method to tell your  
/// custom layout container to place  
its subviews. From this method, call  
/// the  
``LayoutSubview/place(at:anchor:proposal:  
)` method on each  
/// element in `subviews` to tell the  
subviews where to appear in the  
/// user interface.  
///  
/// For example, you can create a  
basic vertical stack that places views  
/// in a column, with views  
horizontally aligned on their leading  
edge:  
///  
/// struct BasicVStack: Layout {  
///     func placeSubviews(  
///         in bounds: CGRect,  
///         proposal:  
ProposedViewSize,  
///         subviews: Subviews,  
///         cache: inout ())  
///     ) {  
///         var point =  
bounds.origin  
///         for subview in  
subviews {  
///             subview.place(at:  
point, anchor: .topLeading,  
proposal: .unspecified)
```

```
    /// point.y +=
subview.dimensions(in: .unspecified).height
    ///
    /// }
    ///
    ///
    /// // This layout also needs
a sizeThatFits() implementation.
    ///
    ///
    /// The example creates a placement
point that starts at the origin of the
    /// specified `bounds` input and uses
that to place the first subview. It
    /// then moves the point in the y
dimension by the subview's height,
    /// which it reads using the
``LayoutSubview/dimensions(in:)`` method.
    /// This prepares the point for the
next iteration of the loop. All
    /// subview operations use an
``ProposedViewSize/unspecified`` size
    /// proposal to indicate that
subviews should use and report their
ideal
    /// size.
    ///
    /// A more complex layout container
might add space between subviews
    /// according to their
``LayoutSubview/spacing`` preferences, or
a
    /// fixed space based on input
```

configuration. For example, you can extend

```
    /// the basic vertical stack's placement method to calculate the
    /// preferred distances between adjacent subviews and store the results in
    /// an array:
    ///
    ///     let spacing: [CGFloat] =
    subviews.indices.dropLast().map { index in
        ///
        subviews[index].spacing.distance(
            ///                         to: subviews[index + 1].spacing,
            ///                         along: .vertical)
        ///
        /// }
        ///
        /// The spacing's
``ViewSpacing/distance(to:along:)`` method considers the
    /// preferences of adjacent views on the edge where they meet. It returns
    /// the smallest distance that satisfies both views' preferences for the
    /// given edge. For example, if one view prefers at least `2` points on its
    /// bottom edge, and the next view prefers at least `8` points on its top
    /// edge, the distance method returns `8`, because that's the smallest
    /// value that satisfies both
```

preferences.

```
///
/// Update the placement calculations
to use the spacing values:
///
///     var point = bounds.origin
///     for (index, subview) in
subviews.enumerated() {
    ///         if index > 0 { point.y +=
spacing[index - 1] } // Add spacing.
    ///         subview.place(at: point,
anchor: .topLeading,
proposal: .unspecified)
    ///         point.y +=
subview.dimensions(in: .unspecified).height
}
///
/// Be sure that you use computations
during placement that are consistent
/// with those in your implementation
of other protocol methods for a given
/// set of inputs. For example, if
you add spacing during placement,
/// make sure your implementation of
///
``sizeThatFits(proposal:subviews:cache:)``
accounts for the extra space.
/// Similarly, if the sizing method
returns different values for different
/// size proposals, make sure the
placement method responds to its
/// `proposal` input in the same way.
```

```
///  
/// - Parameters:  
///   - bounds: The region that the  
container view's parent allocates to the  
///   container view, specified in  
the parent's coordinate space.  
///   Place all the container's  
subviews within the region.  
///   The size of this region  
matches a size that your container  
///   previously returned from a  
call to the  
///  
``sizeThatFits(proposal:subviews:cache:)``  
` method.  
///   - proposal: The size proposal  
from which the container generated the  
///   size that the parent used to  
create the `bounds` parameter.  
///   The parent might propose more  
than one size before calling the  
///   placement method, but it  
always uses one of the proposals and the  
///   corresponding returned size  
when placing the container.  
///   - subviews: A collection of  
proxies that represent the  
///   views that the container  
arranges. Use the proxies in the  
collection  
///   to get information about the  
subviews and to tell the subviews  
///   where to appear.
```

```
    /// - cache: Optional storage for  
calculated data that you can share among  
    /// the methods of your custom  
layout container. See  
    /// ``makeCache(subviews:)`` for  
details.
```

```
func placeSubviews(in bounds: CGRect,  
proposal: ProposedViewSize, subviews:  
Self.Subviews, cache: inout Self.Cache)
```

```
    /// Returns the position of the  
specified horizontal alignment guide  
along
```

```
    /// the x axis.
```

```
    ///
```

```
    /// Implement this method to return a  
value for the specified alignment
```

```
    /// guide of a custom layout  
container. The value you return affects
```

```
    /// the placement of the container as  
a whole, but it doesn't affect how the
```

```
    /// container arranges subviews  
relative to one another.
```

```
    ///
```

```
    /// You can use this method to put an  
alignment guide in a nonstandard
```

```
    /// position. For example, you can  
indent the container's leading edge
```

```
    /// alignment guide by 10 points:
```

```
    ///
```

```
    /// extension BasicVStack {
```

```
    ///     func explicitAlignment(
```

```
    ///         of guide:
```

```
HorizontalAlignment,
    /// in bounds: CGRect,
    /// proposal:
ProposedViewSize,
    /// subviews: Subviews,
    /// cache: inout ()
    /// ) -> CGFloat? {
    /// if guide == .leading
{
    /// return
bounds minX + 10
    /// }
    /// return nil
    /// }
    /// }
    /// The above example returns `nil` for other guides to indicate that they
    /// don't have an explicit value. A guide without an explicit value behaves
    /// as it would for any other view.
If you don't implement the
    /// method, the protocol's default implementation merges the
    /// subviews' guides.
    ///
    /// - Parameters:
    ///   - guide: The
``HorizontalAlignment`` guide that the
method calculates
    ///   the position of.
    ///   - bounds: The region that the
container view's parent allocates to the
```

```
    ///      container view, specified in
the parent's coordinate space.
    /// - proposal: A proposed size for
the container.
    /// - subviews: A collection of
proxy instances that represent the
    ///      views arranged by the
container. You can use the proxies in the
    ///      collection to get information
about the subviews as you determine
    ///      where to place the guide.
    /// - cache: Optional storage for
calculated data that you can share among
    ///      the methods of your custom
layout container. See
    ///      ``makeCache(subviews:)`` for
details.
    ///
    /// - Returns: The guide's position
relative to the `bounds`.
    /// Return `nil` to indicate that
the guide doesn't have an explicit
    /// value.
func explicitAlignment(of guide:
HorizontalAlignment, in bounds: CGRect,
proposal: ProposedViewSize, subviews:
Self.Subviews, cache: inout Self.Cache)
-> CGFloat?

    /// Returns the position of the
specified vertical alignment guide along
    /// the y axis.
    ///
```

```
    /// Implement this method to return a
    value for the specified alignment
    /// guide of a custom layout
    container. The value you return affects
        /// the placement of the container as
    a whole, but it doesn't affect how the
        /// container arranges subviews
    relative to one another.

    ///
    /// You can use this method to put an
    alignment guide in a nonstandard
    /// position. For example, you can
    raise the container's bottom edge
    /// alignment guide by 10 points:
    ///
    /// extension BasicVStack {
    ///     func explicitAlignment(
    ///         of guide:
    VerticalAlignment,
    ///
    ///             in bounds: CGRect,
    ///             proposal:
    ProposedViewSize,
    ///
    ///             subviews: Subviews,
    ///             cache: inout ())
    /// ) -> CGFloat? {
    ///     if guide == .bottom {
    ///         return
    bounds.minY - 10
    ///
    ///     }
    ///     return nil
    ///
    /// }
    ///
    /// }
```

```
    /// The above example returns `nil`  
for other guides to indicate that they  
    /// don't have an explicit value. A  
guide without an explicit value behaves  
    /// as it would for any other view.  
If you don't implement the  
    /// method, the protocol's default  
implementation merges the  
    /// subviews' guides.  
    ///  
    /// - Parameters:  
    ///   - guide: The  
``VerticalAlignment`` guide that the  
method calculates  
    ///   the position of.  
    ///   - bounds: The region that the  
container view's parent allocates to the  
    ///   container view, specified in  
the parent's coordinate space.  
    ///   - proposal: A proposed size for  
the container.  
    ///   - subviews: A collection of  
proxy instances that represent the  
    ///   views arranged by the  
container. You can use the proxies in the  
    ///   collection to get information  
about the subviews as you determine  
    ///   where to place the guide.  
    ///   - cache: Optional storage for  
calculated data that you can share among  
    ///   the methods of your custom  
layout container. See  
    ///   ``makeCache(subviews:)`` for
```

```
details.  
    ///  
    /// - Returns: The guide's position  
    /// relative to the `bounds`.  
    /// Return `nil` to indicate that  
    /// the guide doesn't have an explicit  
    /// value.  
    func explicitAlignment(of guide:  
        VerticalAlignment, in bounds: CGRect,  
        proposal: ProposedViewSize, subviews:  
        Self.Subviews, cache: inout Self.Cache)  
        -> CGFloat?  
    }  
  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension Layout {  
  
    /// The default property values for a  
    /// layout.  
    ///  
    /// If you don't implement the  
    /// ``layoutProperties-5rb5b`` method in  
    /// your custom layout, the protocol  
    /// uses this default implementation  
    /// instead, which returns a  
    /// ``LayoutProperties`` instance with  
    /// default values. The properties  
    /// instance contains information about the  
    /// layout container, like a  
    /// ``LayoutProperties/stackOrientation``  
    /// property that indicates the  
    /// container's major axis.
```

```
    public static var layoutProperties:  
LayoutProperties { get }  
  
        /// Reinitializes a cache to a new  
value.  
        ///  
        /// If you don't implement the  
``updateCache(_:subviews:)`` method in  
        /// your custom layout, the protocol  
uses this default implementation  
        /// instead, which calls  
``makeCache(subviews:)``.  
    public func updateCache(_ cache:  
inout Self.Cache, subviews:  
Self.Subviews)  
  
        /// Returns the result of merging the  
horizontal alignment guides of all  
        /// subviews.  
        ///  
        /// If you don't implement the  
        ///  
``explicitAlignment(of:in:proposal:subvie  
ws:cache:)`` method in  
        /// your custom layout, the protocol  
uses this default implementation  
        /// instead, which merges the guides  
of all the subviews.  
    public func explicitAlignment(of  
guide: HorizontalAlignment, in bounds:  
CGRect, proposal: ProposedViewSize,  
subviews: Self.Subviews, cache: inout  
Self.Cache) -> CGFloat?
```

```
    /// Returns the result of merging the
    vertical alignment guides of all
    /// subviews.
    ///
    /// If you don't implement the
    ///
``explicitAlignment(of:in:proposal:subvie
ws:cache:)`` method in
    // your custom layout, the protocol
uses this default implementation
    // instead, which merges the guides
of all the subviews.
    public func explicitAlignment(of
guide: VerticalAlignment, in bounds:
CGRect, proposal: ProposedViewSize,
subviews: Self.Subviews, cache: inout
Self.Cache) -> CGFloat?

    /// Returns the union of all subview
spacing.
    ///
    /// If you don't implement the
``spacing(subviews:cache:)`` method in
    // your custom layout, the protocol
uses this default implementation
    // instead, which returns the union
of the spacing preferences of all
    // the layout's subviews.
    public func spacing(subviews:
Self.Subviews, cache: inout Self.Cache)
-> ViewSpacing
{
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension Layout where Self.Cache == () {  
  
    /// Returns the empty value when your  
    layout doesn't require a cache.  
    ///  
    /// If you don't implement the  
    ``makeCache(subviews:)`` method in  
    /// your custom layout, the protocol  
    uses this default implementation  
    /// instead, which returns an empty  
    value.  
    public func makeCache(subviews:  
Self.Subviews) -> Self.Cache  
}
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension Layout {  
  
    /// Combines the specified views into  
    a single composite view using  
    /// the layout algorithms of the  
    custom layout container.  
    ///  
    /// Don't call this method directly.  
    SwiftUI calls it when you  
    /// instantiate a custom layout that  
    conforms to the ``Layout``  
    /// protocol:  
    ///
```

```
    ///      BasicVStack { // Implicitly
calls callAsFunction.
    ///          Text("A View")
    ///          Text("Another View")
    ///      }
    ///
    /// For information about how Swift
uses the `callAsFunction()` method to
    /// simplify call site syntax, see
    /// [Methods with Special Names]
(https://docs.swift.org/swift-book/ReferenceManual/Declarations.html#ID622)
    /// in *The Swift Programming
Language*.
    ///
    /// - Parameter content: A
``ViewBuilder`` that contains the views
to
    ///     lay out.
    ///
    /// - Returns: A composite view that
combines all the input views.
public func
callAsFunction<V>(@ViewBuilder _ content:
() -> V) -> some View where V : View
}

/// A direction in which SwiftUI can lay
out content.
///
/// SwiftUI supports both left-to-right
and right-to-left directions
```

```
/// for laying out content to support
different languages and locales.
/// The system sets the value based on
the user's locale, but
/// you can also use the
``View/environment(_:_:)`` modifier
/// to override the direction for a view
and its child views:
///
///     MyView()
///         .environment(\.layoutDirection,
///         .rightToLeft)
///
/// You can also read the
``EnvironmentValues/layoutDirection`` environment
/// value to find out which direction applies to a particular environment.
/// However, in many cases, you don't need to take any action based on this
/// value. SwiftUI horizontally flips the x position of each view within its
/// parent, so layout calculations automatically produce the desired effect
/// for both modes without any changes.
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
public enum LayoutDirection : Hashable,
CaseIterable, Sendable {

    /// A left-to-right layout direction.
    case leftToRight
```

```
/// A right-to-left layout direction.  
case rightToLeft  
  
    /// 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:  
LayoutDirection, b: LayoutDirection) ->  
Bool  
  
    /// 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)
```

```
    /// A type that can represent a
collection of all values of this type.
@available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
public typealias AllCases =
[LayoutDirection]

    /// A collection of all values of
this type.
nonisolated public static var
allCases: [LayoutDirection] { get }
```

```
    /// 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 description of what should happen  
when the layout direction changes.
```

```
///
```

```
/// A `LayoutDirectionBehavior` can be  
used with the `layoutDirectionBehavior`  
/// view modifier or the  
`layoutDirectionBehavior` property of  
`Shape`.
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)
```

```
public enum LayoutDirectionBehavior :  
Hashable, Sendable {
```

```
    /// A behavior that doesn't mirror  
when the layout direction changes.
```

```
    case fixed
```

```
    /// A behavior that mirrors when the  
layout direction has the specified  
    /// value.
```

```
    ///
```

```
    /// If you develop your view or shape  
in an LTR context, you can use
```

```
    /// `.mirrors(in: .rightToLeft)`  
(which is equivalent to ` `.mirrors` ) to  
    /// mirror your content when the  
layout direction is RTL (and keep the  
    /// original version in LTR). If you  
developer in an RTL context, you can  
    /// use ` `.mirrors(in: .leftToRight)`  
to mirror your content when the layout  
    /// direction is LTR (and keep the  
original version in RTL).  
    case mirrors(in: LayoutDirection)  
  
        /// A behavior that mirrors when the  
layout direction is right-to-left.  
    public static var mirrors:  
LayoutDirectionBehavior { get }  
  
        /// 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:
LayoutDirectionBehavior, b:
LayoutDirectionBehavior) -> 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 }  
}
```

```
/// Layout-specific properties of a  
layout container.
```

```
///
```

```
/// This structure contains configuration  
information that's
```

```
/// applicable to a layout container. For  
example, the ``stackOrientation``  
/// value indicates the layout's primary  
axis, if any.
```

```
///
```

```
/// You can use an instance of this type  
to characterize a custom layout
```

```
/// container, which is a type that  
conforms to the ``Layout`` protocol.
```

```
/// Implement the protocol's  
``Layout/layoutProperties-5rb5b``  
property
```

```
/// to return an instance. For example,  
you can indicate that your layout
```

```
/// has a vertical stack orientation:
```

```
///
```

```
///     extension BasicVStack {
///         static var layoutProperties:
LayoutProperties {
///             var properties =
LayoutProperties()
///             properties.stackOrientation = .vertical
///             return properties
///         }
///     }
///     /// If you don't implement the property
///     in your custom layout, the protocol
///     provides a default implementation
///     that returns a `LayoutProperties`
///     instance with default values.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
public struct LayoutProperties : Sendable
{
    /// Creates a default set of
properties.
    ///
    /// Use a layout properties instance
to provide information about
    /// a type that conforms to the
``Layout`` protocol. For example, you
    /// can create a layout properties
instance in your layout's implementation
    /// of the
``Layout/layoutProperties-5rb5b`` method,
and use it to
```

```
    /// indicate that the layout has a
``Axis/vertical`` orientation:
    /**
     * extension BasicVStack {
     *     static var
layoutProperties: LayoutProperties {
    /**
     *         var properties =
LayoutProperties()
    /**
properties.stackOrientation = .vertical
    /**
     *         return properties
    /**
}
    /**
}
    /**
public init()

    /**
 The orientation of the containing
stack-like container.
    /**
    /**
 Certain views alter their
behavior based on the stack orientation
    /**
 of the container that they appear
in. For example, ``Spacer`` and
    /**
 ``Divider`` align their major
axis to match that of their container.
    /**
    /**
 Set the orientation for your
custom layout container by returning a
    /**
 configured ``LayoutProperties``
instance from your ``Layout``
    /**
 type's implementation of the
``Layout/layoutProperties-5rb5b``
    /**
 method. For example, you can
```

```
indicate that your layout has a
    /// ``Axis/vertical`` major axis:
    ///
    ///     extension BasicVStack {
    ///         static var
layoutProperties: LayoutProperties {
    ///             var properties =
LayoutProperties()
    ///
properties.stackOrientation = .vertical
    ///                     return properties
    ///
    ///     }
    ///
    /// A value of `nil`, which is the
default when you don't specify a
    /// value, indicates an unknown
orientation, or that a layout isn't
    /// one-dimensional.
public var stackOrientation: Axis?
}
```

```
/// A proxy that represents one subview
of a layout.
///
/// This type acts as a proxy for a view
that your custom layout container
/// places in the user interface.
``Layout`` protocol methods
/// receive a ``LayoutSubviews``
collection that contains exactly one
/// proxy for each of the subviews
arranged by your container.
```

```
///  
/// Use a proxy to get information about  
the associated subview, like its  
/// dimensions, layout priority, or  
custom layout values. You also use the  
/// proxy to tell its corresponding  
subview where to appear by calling the  
/// proxy's  
``LayoutSubview/place(at:anchor:proposal:  
``` method.  
/// Do this once for each subview from  
your implementation of the layout's  
///  
``Layout/placeSubviews(in:proposal:subvie  
ws:cache:)`` method.  
///  
/// You can read custom layout values  
associated with a subview  
/// by using the property's key as an  
index on the subview. For more  
/// information about defining, setting,  
and reading custom values,  
/// see ``LayoutValueKey``.  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
public struct LayoutSubview : Equatable {  
  
    /// Gets the value for the subview  
that's associated with the specified key.  
    ///  
    /// If you define a custom layout  
value using ``LayoutValueKey``,  
    /// you can read the key's associated
```

```
value for a given subview in
    /// a layout container by indexing
the container's subviews with
    /// the key type. For example, if you
define a `Flexibility` key
    /// type, you can put the associated
values of all the layout's
    /// subviews into an array:
    ///
    ///     let flexibilities =
subviews.map { subview in
    ///             subview[Flexibility.self]
    ///         }
    ///
    /// For more information about
creating a custom layout, see ``Layout``.
public subscript<K>(key: K.Type) ->
K.Value where K : LayoutValueKey { get }

    /// The layout priority of the
subview.
    ///
    /// If you define a custom layout
type using the ``Layout``
    /// protocol, you can read this value
from subviews and use the value
    /// when deciding how to assign space
to subviews. For example, you
    /// can read all of the subview
priorities into an array before
    /// placing the subviews in a custom
layout type called `BasicVStack`:
    ///
```

```
/// extension BasicVStack {
///     func placeSubviews(
///         in bounds: CGRect,
///         proposal:
/// ProposedViewSize,
///         subviews: Subviews,
///         cache: inout ())
///     ) {
///         let priorities =
/// subviews.map { subview in
///             subview.priority
///         }
///         // Place views, based
on priorities.
///     }
/// }
///
/// Set the layout priority for a
view that appears in your layout by
/// applying the
``View/layoutPriority(_:)`` view
modifier. For example,
/// you can assign two different
priorities to views that you arrange
/// with `BasicVStack`:
///
///     BasicVStack {
///         Text("High priority")
///             .layoutPriority(10)
///         Text("Low priority")
///             .layoutPriority(1)
///     }
```

```
///  
public var priority: Double { get }  
  
/// Asks the subview for its size.  
///  
/// Use this method as a convenience  
to get the ``ViewDimensions/width``  
/// and ``ViewDimensions/height``  
properties of the ``ViewDimensions``  
/// instance returned by the  
``dimensions(in:)`` method, reported as a  
///  
<doc://com.apple.documentation/documentation/CoreFoundation/CGSize>  
/// instance.  
///  
/// – Parameter proposal: A proposed  
size for the subview. In SwiftUI,  
/// views choose their own size,  
but can take a size proposal from  
/// their parent view into account  
when doing so.  
///  
/// – Returns: The size that the  
subview chooses for itself, given the  
/// proposal from its container  
view.  
public func sizeThatFits(_ proposal:  
ProposedViewSize) -> CGSize  
  
/// Asks the subview for its  
dimensions and alignment guides.  
///
```

```
    /// Call this method to ask a subview  
of a custom ``Layout`` type  
    /// about its size and alignment  
properties. You can call it from  
    /// your implementation of any of  
that protocol's methods, like  
    ///  
``Layout/placeSubviews(in:proposal:subvie  
ws:cache:)`` or  
    ///  
``Layout/sizeThatFits(proposal:subviews:c  
ache:)``, to get  
    /// information for your layout  
calculations.  
    ///  
    /// When you call this method, you  
propose a size using the `proposal`  
    /// parameter. The subview can choose  
its own size, but might take the  
    /// proposal into account. You can  
call this method more than  
    /// once with different proposals to  
find out if the view is flexible.  
    /// For example, you can propose:  
    ///  
    /// * ``ProposedViewSize/zero`` to  
get the subview's minimum size.  
    /// * ``ProposedViewSize/infinity``  
to get the subview's maximum size.  
    /// *  
``ProposedViewSize/unspecified`` to get  
the subview's ideal size.  
    ///
```

```
    /// If you need only the view's
    /// height and width, you can use the
    /// ``sizeThatFits(_:)`` method
    instead.
    ///
    /// - Parameter proposal: A proposed
    /// size for the subview. In SwiftUI,
    /// views choose their own size,
    /// but can take a size proposal from
    /// their parent view into account
    /// when doing so.
    ///
    /// - Returns: A ``ViewDimensions``
    /// instance that includes a height
    /// and width, as well as a set of
    /// alignment guides.
    public func dimensions(in proposal:
    ProposedViewSize) -> ViewDimensions

    /// The subviews's preferred spacing
    values.
    ///
    /// This ``ViewSpacing`` instance
    indicates how much space a subview
    /// in a custom layout prefers to
    have between it and the next view.
    /// It contains preferences for all
    edges, and might take into account
    /// the type of both this and the
    adjacent view. If your ``Layout`` type
    /// places subviews based on spacing
    preferences, use this instance
    /// to compute a distance between
```

```
this subview and the next. See
    /**
``Layout/placeSubviews(in:proposal:subvie
ws:cache:)`` for an
    /// example.
    /**
    /// You can also merge this instance
with instances from other subviews
    /// to construct a new instance
that's suitable for the subviews'
container.
    /// See
``Layout/spacing(subviews:cache:)``.
    public var spacing: ViewSpacing { get
}

    /// Assigns a position and proposed
size to the subview.
    /**
    /// Call this method from your
implementation of the ``Layout``
    /// protocol's
``Layout/placeSubviews(in:proposal:subvie
ws:cache:)``
    /// method for each subview arranged
by the layout.
    /// Provide a position within the
container's bounds where the subview
    /// should appear, and an anchor that
indicates which part of the subview
    /// appears at that point.
    /**
    /// Include a proposed size that the
```

subview can take into account when

    /// sizing itself. To learn the  
    subview's size for a given proposal  
    before

        /// calling this method, you can call  
        the ``dimensions(in:)`` or

        /// ``sizeThatFits(\_:)`` method on  
        the subview with the same proposal.

        /// That enables you to know subview  
        sizes before committing to subview

        /// positions.

        ///

        /// > Important: Call this method  
        only from within your

        /// ``Layout`` type's  
        implementation of the

        ///

        ``Layout/placeSubviews(in:proposal:subvie  
        ws:cache:)`` method.

        ///

        /// If you call this method more than  
        once for a subview, the last call

        /// takes precedence. If you don't  
        call this method for a subview, the

        /// subview appears at the center of  
        its layout container and uses the

        /// layout container's size proposal.

        ///

        /// - Parameters:

        /// - position: The place where the  
        anchor of the subview should

        /// appear in its container view,  
        relative to container's bounds.

```
    /// - anchor: The unit point on the
    subview that appears at `position`.
    /// You can use a built-in point,
    like ``UnitPoint/center``, or
    /// you can create a custom
    ``UnitPoint``.
    /// - proposal: A proposed size for
    the subview. In SwiftUI,
    /// views choose their own size,
    but can take a size proposal from
    /// their parent view into
    account when doing so.
    public func place(at position:
    CGPoint, anchor: UnitPoint = .topLeading,
    proposal: ProposedViewSize)
```

```
    /// 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:
    LayoutSubview, b: LayoutSubview) -> Bool
}
```

```
/// A collection of proxy values that
```

```
represent the subviews of a layout view.  
///  
/// You receive a `LayoutSubviews` input  
to your implementations of  
/// ``Layout`` protocol methods, like  
///  
``Layout/placeSubviews(in:proposal:subvie  
ws:cache:)`` and  
///  
``Layout/sizeThatFits(proposal:subviews:c  
ache:)``. The `subviews`  
/// parameter (which the protocol aliases  
to the ``Layout/Subviews`` type)  
/// is a collection that contains proxies  
for the layout's subviews (of type  
/// ``LayoutSubview``). The proxies  
appear in the collection in the same  
/// order that they appear in the  
``ViewBuilder`` input to the layout  
/// container. Use the proxies to perform  
layout operations.  
///  
/// Access the proxies in the collection  
as you would the contents of any  
/// Swift random-access collection. For  
example, you can enumerate all of the  
/// subviews and their indices to inspect  
or operate on them:  
///  
///     for (index, subview) in  
subviews.enumerated() {  
///         // ...  
///     }
```

```
///  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
public struct LayoutSubviews : Equatable,  
RandomAccessCollection, Sendable {  
  
    /// A type that contains a  
    subsequence of proxy values.  
    public typealias SubSequence =  
LayoutSubviews  
  
    /// A type that contains a proxy  
    value.  
    public typealias Element =  
LayoutSubview  
  
    /// A type that you can use to index  
    proxy values.  
    public typealias Index = Int  
  
    /// The layout direction inherited by  
    the container view.  
    ///  
    /// SwiftUI supports both left-to-  
    right and right-to-left directions.  
    /// Read this property within a  
    custom layout container  
    /// to find out which environment the  
    container is in.  
    ///  
    /// In most cases, you don't need to  
    take any action based on this  
    /// value. SwiftUI horizontally flips
```

```
the x position of each view within its
    /// parent when the mode switches, so
layout calculations automatically
    /// produce the desired effect for
both directions.
    public var layoutDirection:
LayoutDirection

    /// The index of the first subview.
    public var startIndex: Int { get }

    /// An index that's one higher than
the last subview.
    public var endIndex: Int { get }

    /// Gets the subview proxy at a
specified index.
    public subscript(index: Int) ->
LayoutSubviews.Element { get }

    /// Gets the subview proxies in the
specified range.
    public subscript(bounds: Range<Int>)
-> LayoutSubviews { get }

    /// Gets the subview proxies with the
specified indicies.
    public subscript<S>(indices: S) ->
LayoutSubviews where S : Sequence,
S.Element == Int { get }

    /// 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 == (lhs:  
LayoutSubviews, rhs: LayoutSubviews) ->  
Bool
```

```
/// A type that represents the  
indices that are valid for subscripting  
the  
/// collection, in ascending order.  
@available(iOS 16.0, tvOS 16.0,  
watchOS 9.0, macOS 13.0, *)  
public typealias Indices =  
Range<LayoutSubviews.Index>  
  
/// A type that provides the  
collection's iteration interface and  
/// encapsulates its iteration state.  
///  
/// By default, a collection conforms  
to the `Sequence` protocol by  
/// supplying `IndexingIterator` as  
its associated `Iterator`  
/// type.  
@available(iOS 16.0, tvOS 16.0,
```

```
watchOS 9.0, macOS 13.0, *)
    public typealias Iterator =
IndexingIterator<LayoutSubviews>
}

/// A key for accessing a layout value of
/// a layout container's subviews.
///
/// If you create a custom layout by
/// defining a type that conforms to the
/// ``Layout`` protocol, you can also
/// create custom layout values
/// that you set on individual views, and
/// that your container view can access
/// to guide its layout behavior. Your
/// custom values resemble
/// the built-in layout values that you
/// set with view modifiers
/// like ``View/layoutPriority(_:)`` and
/// ``View/zIndex(_:)``, but have a
/// purpose that you define.
///
/// To create a custom layout value,
/// define a type that conforms to the
/// ``LayoutValueKey`` protocol and
/// implement the one required property that
/// returns the default value of the
/// property. For example, you can create
/// a property that defines an amount of
/// flexibility for a view, defined as
/// an optional floating point number
/// with a default value of `nil`:
///
```

```
///     private struct Flexibility:  
LayoutValueKey {  
///         static let defaultValue:  
CGFloat? = nil  
///     }  
///  
/// The Swift compiler infers this  
particular key's associated type as an  
/// optional  
<doc://com.apple.documentation/documentation/CoreFoundation/CGFloat>  
/// from this definition.  
///  
/// Set a value on a view  
///  
/// Set the value on a view by adding the  
``View/layoutValue(key:value:)``  
/// view modifier to the view. To make  
your custom value easier to work  
/// with, you can do this in a  
convenience modifier in an extension of  
the  
/// ``View`` protocol:  
///  
///     extension View {  
///         func layoutFlexibility(_  
value: CGFloat?) -> some View {  
///             layoutValue(key:  
Flexibility.self, value: value)  
///         }  
///     }  
///  
/// Use your modifier to set the value on
```

```
any views that need a nondefault
/// value:
///
///     BasicVStack {
///         Text("One View")
///         Text("Another View")
///             .layoutFlexibility(3)
///     }
///
/// Any view that you don't explicitly
set a value for uses the default
/// value, as with the first ``Text``
view, above.
///
/// ### Retrieve a value during layout
///
/// Access a custom layout value using
the key as an index
/// on subview's proxy (an instance of
``LayoutSubview``)
/// and use the value to make decisions
about sizing, placement, or other
/// layout operations. For example, you
might read the flexibility value
/// in your layout view's
``LayoutSubview/sizeThatFits(_:)``  
method, and
/// adjust your size calculations
accordingly:
///
///     extension BasicVStack {
///         func sizeThatFits(
///             proposal:
```

```
ProposedViewSize,
///           subviews: Subviews,
///           cache: inout Void
///     ) -> CGSize {
///
///           // Map the flexibility
///           // property of each subview into an array.
///           let flexibilities =
///           subviews.map { subview in
///           subview[Flexibility.self]
///           }
///
///           // Calculate and return
///           // the size of the layout container.
///           // ...
///           }
///           }
///
/// @available(iOS 16.0, macOS 13.0, tvOS
/// 16.0, watchOS 9.0, *)
public protocol LayoutValueKey {
    /// The type of the key's value.
    ///
    /// Swift typically infers this type
    /// from your implementation of the
    /// ``defaultValue`` property, so you
    /// don't have to define it explicitly.
    associatedtype Value
    /// The default value of the key.
    ///
}
```

```
    /// Implement the `defaultValue`  
    property for a type that conforms to the  
    /// ``LayoutValueKey`` protocol. For  
example, you can create a `Flexibility`  
    /// layout value that defaults to  
`nil`:  
    ////  
    ///     private struct Flexibility:  
LayoutValueKey {  
    ///         static let defaultValue:  
CGFloat? = nil  
    ///     }  
    ////  
    /// The type that you declare for the  
`defaultValue` sets the layout  
    /// key's ``Value`` associated type.  
The Swift compiler infers the key's  
    /// associated type in the above  
example as an optional  
    ////  
<doc://com.apple.documentation/documentation/CoreFoundation/CGFloat>  
    ////  
    /// Any view that you don't  
explicitly set a value for uses the  
default  
    /// value. Override the default value  
for a view using the  
    /// ``View/layoutValue(key:value:)``  
modifier.  
        static var defaultValue: Self.Value {  
get }  
}
```

```
/// The Accessibility Bold Text user
setting options.
///
/// The app can't override the user's
choice before iOS 16, tvOS 16 or
/// watchOS 9.0.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public enum LegibilityWeight : Hashable,
Sendable {

    /// Use regular font weight (no
Accessibility Bold).
    case regular

    /// Use heavier font weight (force
Accessibility Bold).
    case bold

    /// 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:
```

```
LegibilityWeight, b: LegibilityWeight) ->
Bool
```

```
    /// 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)
```

```
    /// 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 linear gradient.
```

```
///
```

```
/// The gradient applies the color
function along an axis, as defined by its
/// start and end points. The gradient
maps the unit space points into the
/// bounding rectangle of each shape
filled with the gradient.
```

```
///
```

```
/// When using a linear gradient as a
shape style, you can also use
```

```
///
```

```
`ShapeStyle/linearGradient(_:startPoint:
endPoint:)``.
```

```
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
```

```
@frozen public struct LinearGradient :
ShapeStyle, View, Sendable {
```

```
    /// Creates a linear gradient from a
    base gradient.
    public init(gradient: Gradient,
startPoint: UnitPoint, endPoint:
UnitPoint)

    /// Creates a linear gradient from a
collection of colors.
    public init(colors: [Color],
startPoint: UnitPoint, endPoint:
UnitPoint)

    /// Creates a linear gradient from a
collection of color stops.
    public init(stops: [Gradient.Stop],
startPoint: UnitPoint, endPoint:
UnitPoint)

    /// The type of view representing the
body of this view.
    /**
     * When you create a custom view,
     Swift infers this type from your
     * implementation of the required
     ``View/body-swift.property`` property.
     */
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Body

    /// The type of shape style this will
resolve to.
    /**
```

```
    /// When you create a custom shape
    style, Swift infers this type
    /// from your implementation of the
    required `resolve` function.
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Resolved = Never
}

/// A keyframe that uses simple linear
interpolation.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct LinearKeyframe<Value> :
KeyframeTrackContent where Value :
Animatable {

    /// Creates a new keyframe using the
    given value and timestamp.
    ///
    /// - Parameters:
    ///   - to: The value of the
    keyframe.
    ///   - duration: The duration of the
    segment defined by this keyframe.
    ///   - timingCurve: A unit curve
    that controls the speed of interpolation.
    public init(_ to: Value, duration:
TimeInterval, timingCurve: UnitCurve
= .linear)

    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
```

```
    public typealias Body =  
LinearKeyframe<Value>  
}  
  
/// The local coordinate space of the  
current view.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
public struct LocalCoordinateSpace :  
CoordinateSpaceProtocol {  
  
    public init()  
  
        /// The resolved coordinate space.  
        public var coordinateSpace:  
CoordinateSpace { get }  
}  
  
/// The key used to look up an entry in a  
strings file or strings dictionary  
/// file.  
///  
/// Initializers for several SwiftUI  
types -- such as ``Text``, ``Toggle``,  
/// ``Picker`` and others -- implicitly  
look up a localized string when you  
/// provide a string literal. When you  
use the initializer `Text("Hello")`,  
/// SwiftUI creates a  
`LocalizedStringKey` for you and uses  
that to look up a  
/// localization of the `Hello` string.  
This works because `LocalizedStringKey`
```

```
/// conforms to
///
<doc://com.apple.documentation/documentation/Swift/ExpressibleByStringLiteral>.
///
/// Types whose initializers take a
`LocalizedStringKey` usually have
/// a corresponding initializer that
accepts a parameter that conforms to
///
<doc://com.apple.documentation/documentation/Swift/StringProtocol>. Passing
/// a `String` variable to these
initializers avoids localization, which
is
/// usually appropriate when the variable
contains a user-provided value.
///
/// As a general rule, use a string
literal argument when you want
/// localization, and a string variable
argument when you don't. In the case
/// where you want to localize the value
of a string variable, use the string to
/// create a new `LocalizedStringKey`
instance.
///
/// The following example shows how to
create ``Text`` instances both
/// with and without localization. The
title parameter provided to the
/// ``Section`` is a literal string, so
SwiftUI creates a
```

```
/// `LocalizedStringKey` for it. However,  
the string entries in the  
/// `messageStore.today` array are  
`String` variables, so the ``Text`` views  
/// in the list use the string values  
verbatim.  
///  
///     List {  
///         Section(header:  
Text("Today")) {  
///             ForEach(messageStore.today) { message in  
///                 Text(message.title)  
///             }  
///         }  
///     }  
///  
/// If the app is localized into Japanese  
with the following  
/// translation of its  
`Localizable.strings` file:  
///  
///     ``other  
///     "Today" = "今日";  
///     ``  
///  
/// When run in Japanese, the example  
produces a  
/// list like the following, localizing  
"Today" for the section header, but not  
/// the list items.  
///  
/// ![A list with a single section header
```

```
displayed in Japanese.  
/// The items in the list are all in  
English: New for Monday, Account update,  
/// and Server  
/// maintenance.](SwiftUI-  
LocalizedStringKey-Today-List-  
Japanese.png)  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct  
LocalizedStringKey : Equatable,  
ExpressibleByStringInterpolation {  
  
    /// Creates a localized string key  
from the given string value.  
    ///  
    /// - Parameter value: The string to  
use as a localization key.  
    public init(_ value: String)  
  
    /// Creates a localized string key  
from the given string literal.  
    ///  
    /// - Parameter value: The string  
literal to use as a localization key.  
    public init(stringLiteral value:  
String)  
  
    /// Creates a localized string key  
from the given string interpolation.  
    ///  
    /// To create a localized string key  
from a string interpolation, use
```

```
    /// the `\\()` string interpolation
syntax. Swift matches the parameter
    /// types in the expression to one of
the `appendInterpolation` methods
    /// in
``LocalizedStringKey/StringInterpolation``
. The interpolated
    /// types can include numeric values,
Foundation types, and SwiftUI
    /// ``Text`` and ``Image`` instances.
    ///
    /// The following example uses a
string interpolation with two arguments:
    /// an unlabeled
    ///
<doc://com.apple.documentation/documentation/Foundation/Date>
    /// and a ``Text/DateStyle`` labeled
`style`. The compiler maps these to the
    /// method
    ///
``LocalizedStringKey/StringInterpolation/
appendInterpolation(_:_:)``
    /// as it builds the string that it
creates the
    /// ``LocalizedStringKey`` with.
    ///
    ///     let key =
LocalizedStringKey("Date is \
(company.foundedDate, style: .offset)")
    ///     let text = Text(key) // Text
contains "Date is +45 years"
    ///
```

```
    /// You can write this example more
    // concisely, implicitly creating a
    /// ``LocalizedStringKey`` as the
    parameter to the ``Text``
    /// initializer:
    /**
     ///     let text = Text("Date is \
    (company.foundedDate, style: .offset)")
    /**
     /// - Parameter stringInterpolation:
    The string interpolation to use as the
    /// localization key.
    public init(stringInterpolation:
    LocalizedStringKey.StringInterpolation)
```

```
    /// Represents the contents of a
    string literal with interpolations
    /// while it's being built, for use
    in creating a localized string key.
    public struct StringInterpolation :  
StringInterpolationProtocol {  
  
        /// Creates an empty instance
    ready to be filled with string literal
    content.
        /**
         /// Don't call this initializer
    directly. Instead, initialize a variable
    or
        /// constant using a string
    literal with interpolated expressions.
        /**
         /// Swift passes this initializer
```

a pair of arguments specifying the size of

    /// the literal segments and the number of interpolated segments. Use this  
    /// information to estimate the amount of storage you will need.

    ///

    /// - Parameter literalCapacity:

The approximate size of all literal segments

    /// combined. This is meant to be passed to

`String.reserveCapacity(\_:)`;

    /// it may be slightly larger or smaller than the sum of the counts of each

    /// literal segment.

    /// - Parameter

interpolationCount: The number of interpolations which will be

    /// appended. Use this value to estimate how much additional capacity will

    /// be needed for the interpolated segments.

**public init**(literalCapacity: Int,  
interpolationCount: Int)

    /// Appends a literal string.

    ///

    /// Don't call this method directly; it's used by the compiler when

    /// interpreting string

```
interpolations.  
    ///  
    /// - Parameter literal: The  
literal string to append.  
    public mutating func  
appendLiteral(_ literal: String)  
  
        /// Appends a literal string  
segment to a string interpolation.  
        ///  
        /// Don't call this method  
directly; it's used by the compiler when  
        /// interpreting string  
interpolations.  
        ///  
        /// - Parameter string: The  
literal string to append.  
    public mutating func  
appendInterpolation(_ string: String)  
  
        /// Appends an optionally-  
formatted instance of a Foundation type  
        /// to a string interpolation.  
        ///  
        /// Don't call this method  
directly; it's used by the compiler when  
        /// interpreting string  
interpolations.  
        ///  
        /// - Parameters:  
        ///     - subject: The Foundation  
object to append.  
        ///     - formatter: A formatter to
```

```
convert `subject` to a string
    ///      representation.
    public mutating func
appendInterpolation<Subject>(_ subject:
Subject, formatter: Formatter? = nil)
where Subject : ReferenceConvertible

    /// Appends an optionally-
formatted instance of an Objective-C
subclass
    /// to a string interpolation.
    ///
    /// Don't call this method
directly; it's used by the compiler when
    /// interpreting string
interpolations.
    ///
    /// The following example shows
how to use a
    ///
<doc://com.apple.documentation/documentation/Foundation/Measurement>
    /// value and a
    ///
<doc://com.apple.documentation/documentation/Foundation/MeasurementFormatter>
    /// to create a
``LocalizedStringKey`` that uses the
formatter
    /// style
    ///
<doc://com.apple.documentation/documentation/foundation/Formatter/UnitStyle/long>
```

```
    /// when generating the
measurement's string representation.
Rather than
    /// calling
`appendInterpolation(_:formatter)`
directly, the code
    /// gets the formatting behavior
implicitly by using the `\\()`
    /// string interpolation syntax.
    ///
    ///     let siResistance =
Measurement(value: 640, unit:
UnitElectricResistance.ohms)
    ///     let formatter =
MeasurementFormatter()
    ///     formatter.unitStyle
= .long
    ///     let key =
LocalizedStringKey ("Resistance: \
(siResistance, formatter: formatter)")
    ///     let text1 = Text(key) //
Text contains "Resistance: 640 ohms"
    ///
    /// - Parameters:
    ///     - subject: An
<doc://com.apple.documentation/documentation/objectivec/NSObject>
    ///     to append.
    ///     - formatter: A formatter to
convert `subject` to a string
    ///     representation.
public mutating func
appendInterpolation<Subject>(_ subject:
```

```
Subject, formatter: Formatter? = nil)
where Subject : NSObject
```

```
    /// Appends the formatted
    representation of a nonstring type
    /// supported by a corresponding
    format style.
    ///
    /// Don't call this method
    directly; it's used by the compiler when
    /// interpreting string
    interpolations.
    ///
    /// The following example shows
    how to use a string interpolation to
    /// format a
    ///
<doc://com.apple.documentation/documentation/Foundation/Date>
    /// with a
    ///
<doc://com.apple.documentation/documentation/Foundation/DateFormatStyle> and
    /// append it to static text. The
    resulting interpolation implicitly
    /// creates a
``LocalizedStringKey``, which a ``Text``
uses to provide
    /// its content.
    ///
    ///     Text("The time is \
(myDate, format:
Date.FormatStyle(date: .omitted,
```

```
time:.complete))")
    /**
     /**
      - Parameters:
        /**
         - input: The instance to
format and append.
        /**
         - format: A format style to
use when converting `input` into a string
        /**
         representation.
        @available(iOS 15.0, macOS 12.0,
tvOS 15.0, watchOS 8.0, *)
        public mutating func
appendInterpolation<F>(_ input:
F.FormatInput, format: F) where F :
FormatStyle, F.FormatInput : Equatable,
F.FormatOutput == String

        /**
         Appends the formatted
representation of a nonstring type
        /**
         supported by a corresponding
format style.
        /**
        /**
         Don't call this method
directly; it's used by the compiler when
        /**
         interpreting string
interpolations.
        /**
        /**
         The following example shows
how to use a string interpolation to
        /**
         format a
        /**
<doc://com.apple.documentation/documentation/Foundation/Date>
        /**
         with a
```

```
    ///
<doc://com.apple.documentation/documentation/Foundation/DateFormatStyle> and
    /// append it to static text. The
resulting interpolation implicitly
    /// creates a
``LocalizedStringKey``, which a ``Text``
uses to provide
    /// its content.
    ///
    ///     Text("The time is \
(myDate, format:
Date.FormatStyle(date: .omitted,
time:.complete).attributedStyle)")
    ///
    /// - Parameters:
    ///   - input: The instance to
format and append.
    ///   - format: A format style to
use when converting `input` into an
attributed
    ///   string representation.
@available(iOS 18.0, macOS 15.0,
tvOS 18.0, watchOS 11.0, visionOS 2.0, *)
public mutating func
appendInterpolation<F>(_ input:
F.FormatInput, format: F) where F :
FormatStyle, F.FormatInput : Equatable,
F.FormatOutput == AttributedString

    /// Appends a type, convertible
to a string by using a default format
    /// specifier, to a string
```

```
interpolation.  
    ///  
    /// Don't call this method  
directly; it's used by the compiler when  
    /// interpreting string  
interpolations.  
    ///  
    /// - Parameters:  
    ///   - value: A primitive type  
to append, such as  
    ///  
<doc://com.apple.documentation/documentation/swift/Int>,  
    ///  
<doc://com.apple.documentation/documentation/swift/UInt32>, or  
    ///  
<doc://com.apple.documentation/documentation/swift/Double>.  
    public mutating func  
appendInterpolation<T>(_ value: T) where  
T : _FormatSpecifiable  
    /// Appends a type, convertible  
to a string with a format specifier,  
    /// to a string interpolation.  
    ///  
    /// Don't call this method  
directly; it's used by the compiler when  
    /// interpreting string  
interpolations.  
    ///  
    /// - Parameters:
```

```
        /// - value: The value to
append.
        /// - specifier: A format
specifier to convert `subject` to a
string
        ///     representation, like `%.f`
for a
        ///
<doc://com.apple.documentation/documentation/swift/Double>, or
        ///     `%.x` to create a
hexidecimal representation of a
        ///
<doc://com.apple.documentation/documentation/swift/UInt32>. For a
        ///     list of available
specifier strings, see
        ///     [String Format Specifiers]
(https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFStrings/formatSpecifiers.html#/apple\_ref/doc/uid/TP40004265).
    public mutating func
appendInterpolation<T>(_ value: T,
specifier: String) where T :
_FormatSpecifiable

        /// Appends the string displayed
by a text view to a string
        /// interpolation.
        ///
        /// Don't call this method
```

```
directly; it's used by the compiler when
    /// interpreting string
interpolations.
```

```
    ///
    /// - Parameters:
    ///   - value: A ``Text``
instance to append.
    @available(iOS 14.0, macOS 11.0,
tvOS 14.0, watchOS 7.0, *)
    public mutating func
appendInterpolation(_ text: Text)
```

```
        /// Appends an attributed string
to a string interpolation.
```

```
        ///
        /// Don't call this method
directly; it's used by the compiler when
    /// interpreting string
interpolations.
```

```
        ///
        /// The following example shows
how to use a string interpolation to
    /// format an
    ///
<doc://com.apple.documentation/documentation/Foundation/AttributedString>
```

```
        /// and append it to static text.
The resulting interpolation implicitly
    /// creates a
``LocalizedStringKey``, which a ``Text``
view uses to provide
    /// its content.
    ///

```

```
    /// struct ContentView: View
{
    /**
     * var nextDate: AttributedString {
     *     /**
     *         var result =
     *     Calendar.current
     *         /**
     *             .nextWeekend(
     * startingAfter: Date.now) !
     *             /**
     *             /**
     *             /**
     *             /**
     *         ide)
     *             /**
     *             /**
     *         ed
     *             /**
     *             )
     *             /**
     *         result.backgroundColor = .green
     *             /**
     *         result.foregroundColor = .white
     *             /**
     *                 return result
     *             /**
     *             /**
     *                 var body: some View {
     *                     /**
     *                         Text("Our next
     * catch-up is on \((nextDate)!)")
     *                     /**
     *                 }
     *             /**
     *             /**
     *                 /**
     * For this example, assume that
     * the app runs on a device set to a
```

```
    /// Russian locale, and has the
following entry in a Russian-localized
    /// `Localizable.strings` file:
    ///
    ///      "Our next catch-up is on
%@!" = "Наша следующая встреча состоится
%@!";
    ///
    /// The attributed string
`nextDate` replaces the format specifier
    /// `%@", maintaining its color
and date-formatting attributes, when
    /// the ``Text`` view renders its
contents:
```

```
    ///
    /// ![A text view with Russian
text, ending with a date that uses white
    /// text on a green
    /// background.]
```

(LocalizedStringKey-AttributedString-Russian)

```
    ///
    /// - Parameter attributedString:
The attributed string to append.
    @available(macOS 12.0, iOS 15.0,
tvOS 15.0, watchOS 8.0, *)
    public mutating func
appendInterpolation(_ attributedString:
AttributedString)
```

/// The type that should be used
for literal segments.

```
    @available(iOS 13.0, tvOS 13.0,
```

```
watchOS 6.0, macOS 10.15, *)
    public typealias
StringLiteralType = String
}

    /// 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:
LocalizedLocalizedStringKey, b:
LocalizedLocalizedStringKey) -> Bool

    /// A type that represents an
extended grapheme cluster literal.
///
/// Valid types for
`ExtendedGraphemeClusterLiteralType` are
`Character`,
/// `String`, and `StaticString`.
@available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias
ExtendedGraphemeClusterLiteralType =
String
```

```
    /// A type that represents a string
literal.
    /**
     /// Valid types for
`StringLiteralType` are `String` and
`StaticString`.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias StringLiteralType =
String

    /// A type that represents a Unicode
scalar literal.
    /**
     /// Valid types for
`UnicodeScalarLiteralType` are
`Unicode.Scalar`,
     /// `Character`, `String`, and
`StaticString`.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias
UnicodeScalarLiteralType = String
}

@available(iOS 16.0, macOS 13, tvOS 16.0,
watchOS 9.0, *)
extension
LocalizedStringKey.StringInterpolation {

    /// Appends the localized string
resource to a string interpolation.
```

```
///  
/// Don't call this method directly;  
it's used by the compiler when  
    /// interpreting string  
interpolations.  
///  
/// - Parameters:  
    /// - value: The localized string  
resource to append.  
    @available(iOS 16.0, macOS 13, tvOS  
16.0, watchOS 9.0, *)  
    public mutating func  
appendInterpolation(_ resource:  
LocalizedStringResource)  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension  
LocalizedStringKey.StringInterpolation {  
  
    /// Appends a localized description  
of a color for accessibility  
    /// to a string interpolation.  
    ///  
    /// Don't call this method directly;  
it's used by the compiler when  
    /// interpreting string  
interpolations.  
    ///  
    /// - Parameters:  
    /// - color: the color being  
described.
```

```
    public mutating func
appendInterpolation(accessibilityName
color: Color)
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension
LocalizedStringKey.StringInterpolation {

    /// Appends an image to a string
interpolation.
    ///
    /// Don't call this method directly;
it's used by the compiler when
    /// interpreting string
interpolations.
    ///
    /// - Parameter image: The image to
append.

    public mutating func
appendInterpolation(_ image: Image)
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension
LocalizedStringKey.StringInterpolation {

    /// Appends a formatted date to a
string interpolation.
    ///
    /// Don't call this method directly;
```

```
it's used by the compiler when
    /// interpreting string
interpolations.
    /**
     /// - Parameters:
     ///   - date: The date to append.
     ///   - style: A predefined style to
format the date with.
    public mutating func
appendInterpolation(_ date: Date, style:
Text.DateStyle)

    /// Appends a date range to a string
interpolation.
    /**
     /// Don't call this method directly;
it's used by the compiler when
    /// interpreting string
interpolations.
    /**
     /// - Parameter dates: The closed
range of dates to append.
    public mutating func
appendInterpolation(_ dates:
ClosedRange<Date>)

    /// Appends a date interval to a
string interpolation.
    /**
     /// Don't call this method directly;
it's used by the compiler when
    /// interpreting string
interpolations.
```

```
    /**
     * - Parameter interval: The date
     * interval to append.
     */
    public mutating func
    appendInterpolation(_ interval:
    DateInterval)
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension
LocalizedStringKey.StringInterpolation {

    /**
     * Appends a timer interval to a
     * string interpolation.
     */
    /**
     * Don't call this method directly;
     * it's used by the compiler when
     * interpreting string
     * interpolations.
     */
    /**
     * - Parameters:
     *   - timerInterval: The interval
     * between where to run the timer.
     *   - pauseTime: If present, the
     * date at which to pause the timer.
     *   - The default is `nil`
     * which indicates to never pause.
     *   - countsDown: Whether to
     * count up or down. The default is `true`.
     *   - showsHours: Whether to
     * include an hours component if there are
     *   - more than 60 minutes left
    
```

on the timer. The default is `true`.

```
    public mutating func  
appendInterpolation(timerInterval:  
ClosedRange<Date>, pauseTime: Date? =  
nil, countsDown: Bool = true, showsHours:  
Bool = true)  
}
```

**extension**

```
LocalizedStringKey.StringInterpolation {
```

```
    /// Appends a text view that displays  
the current system time as defined by the  
    /// given format style, keeping the  
text up to date as time progresses.
```

```
    ///
```

```
    /// Use this initializer to create a  
text view that updates as time  
progresses, just
```

```
    /// like ``init(_:style:)``, but with  
the flexibility of Foundation's  
`FormatStyle`
```

```
    /// protocol.
```

```
    ///
```

```
    /// In the following example, the  
first ``Text`` view presents the offset  
to
```

```
    /// `startDate`, whereas the second  
view displays a stopwatch counting from  
    /// `startDate`. Both views are kept  
up to date as time progresses.
```

```
    ///
```

```
    ///     Text(.currentDate,
```

```
format: .offset(to: startDate))
    ///      Text(.currentDate,
format: .stopwatch(startingAt:
startDate))
    ///
    /// - Note: Don't call this method
directly; it's used by the compiler when
    /// interpreting string
interpolations.
    ///
    /// ## Redaction for Reduced
Luminance
    ///
    /// When the text is displayed with
reduced luminance and frame rate, it
    /// automatically modifies the
`format` or its output so it never shows
outdated
    /// information.
    ///
    /// If the `format` is known to
SwiftUI and allows removing units or
fields, SwiftUI
    /// removes parts that change more
frequently than the frame rate allows.
E.g. a
    /// string like _13 minutes, 22
seconds_ would change to just `13
minutes`.
    ///
    /// Otherwise, SwiftUI inspects the
`durationField`, `dateField`, and
`measurement`
```

```
    /// attributes on the formatted
    output to determine which ranges need to
    be
    /// redacted. If these attributes are
    not present, all digits are redacted
    using
    /// dashes.
    @available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public mutating func
appendInterpolation<V, F>(_ source:
TimeDataSource<V>, format: F) where V ==
F.FormatInput, F : DiscreteFormatStyle,
F.FormatOutput == AttributedString

    /// Appends a text view that displays
the current system time as defined by the
    /// given format style, keeping the
text up to date as time progresses.
    ///
    /// Use this initializer to create a
text view that updates as time
progresses, just
    /// like ``init(_:style:)``, but with
the flexibility of Foundation's
`FormatStyle`
    /// protocol.
    ///
    /// In the following example, the
first ``Text`` view presents the current
date and
    /// time, whereas the second view
displays a soccer clock counting from
```

```
`startDate`.  
    /// Both views are kept up to date as  
time progresses.  
    ///  
    ///     Text(.currentDate,  
format: .dateTime)  
    ///     Text(.durationOffset(to:  
startDate),  
format: .time(pattern: .minuteSecond))  
    ///  
    /// – Note: Don't call this method  
directly; it's used by the compiler when  
    /// interpreting string  
interpolations.  
    ///  
    /// ## Redaction for Reduced  
Luminance  
    ///  
    /// When the text is displayed with  
reduced luminance and frame rate, it  
    /// automatically modifies the  
`format` or its output so it never shows  
outdated  
    /// information.  
    ///  
    /// If the `format` is known to  
SwiftUI and allows removing units or  
fields, SwiftUI  
    /// removes parts that change more  
frequently than the frame rate allows.  
E.g. a  
    /// string like _13 minutes, 22  
seconds_ would change to just `13
```

```
minutes`.  
    ///  
    /// Otherwise, all digits in the  
    // formatted output are redacted using  
    // dashes.  
    @available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
    public mutating func  
appendInterpolation<V, F>(_ source:  
TimeDataSource<V>, format: F) where V ==  
F.FormatInput, F : DiscreteFormatStyle,  
F.FormatOutput == String  
}  
  
/// A set of view properties that may be  
// synchronized between views  
/// using the  
`View.matchedGeometryEffect()` function.  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
@frozen public struct  
MatchedGeometryProperties : OptionSet {  
  
    /// The corresponding value of the  
raw type.  
    ///  
    /// A new instance initialized with  
`rawValue` will be equivalent to this  
    /// instance. For example:  
    ///  
    ///     enum PaperSize: String {  
    ///         case A4, A5, Letter,  
Legal
```

```
    /**
     */
    /**
     *      let selectedSize =
PaperSize.Letter
    /**
     *      print(selectedSize.rawValue)
    /**
     *      // Prints "Letter"
    /**
     *      print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    /**
     *      // Prints "true"
public let rawValue: UInt32

    /**
     * Creates a new option set from the
given raw value.
    /**
    /**
     * This initializer always succeeds,
even if the value passed as `rawValue`
    /**
     * exceeds the static properties
declared as part of the option set. This
    /**
     * example creates an instance of
`ShippingOptions` with a raw value beyond
    /**
     * the highest element, with a bit
mask that effectively contains all the
    /**
     * declared static members.
    /**
    /**
     *      let extraOptions =
ShippingOptions(rawValue: 255)
    /**
print(extraOptions.isStrictSuperset(of: .
all))
    /**
     *      // Prints "true"
    /**

```

```
    /// - Parameter rawValue: The raw  
    value of the option set to create. Each  
    bit  
        /// of `rawValue` potentially  
    represents an element of the option set,  
        /// though raw values may include  
    bits that are not defined as distinct  
        /// values of the `OptionSet` type.  
    @inlinable public init(rawValue:  
UInt32)  
  
        /// The view's position, in window  
    coordinates.  
    public static let position:  
MatchedGeometryProperties  
  
        /// The view's size, in local  
    coordinates.  
    public static let size:  
MatchedGeometryProperties  
  
        /// Both the `position` and `size`  
    properties.  
    public static let frame:  
MatchedGeometryProperties  
  
        /// The type of the elements of an  
    array literal.  
    @available(iOS 14.0, tvOS 14.0,  
watchOS 7.0, macOS 11.0, *)  
    public typealias ArrayLiteralElement  
= MatchedGeometryProperties
```

```
    /// The element type of the option
    set.
    /**
     /// To inherit all the default
     implementations from the `OptionSet`
     protocol,
     /// the `Element` type must be
     `Self`, the default.
     @available(iOS 14.0, tvOS 14.0,
     watchOS 7.0, macOS 11.0, *)
     public typealias Element =
     MatchedGeometryProperties

    /// The raw type that can be used to
    represent all values of the conforming
    /// type.
    /**
     /// Every distinct value of the
     conforming type has a corresponding
     unique
     /// value of the `RawValue` type, but
     there may be values of the `RawValue`-
     /// type that don't have a
     corresponding value of the conforming
     type.
     @available(iOS 14.0, tvOS 14.0,
     watchOS 7.0, macOS 11.0, *)
     public typealias RawValue = UInt32
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension MatchedGeometryProperties :
```

```
Sendable {  
}  
  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
extension MatchedGeometryProperties :  
BitwiseCopyable {  
}  
  
/// A background material type.  
///  
/// You can apply a blur effect to a view  
that appears behind another view by  
/// adding a material with the  
``View/background(_:ignoresSafeAreaEdges:  
)`  
/// modifier:  
///  
///     ZStack {  
///         Color.teal  
///         Label("Flag", systemImage:  
"flag.fill")  
///             .padding()  
///             .background(.regularMater  
ial)  
///     }  
///  
/// In the example above, the ``ZStack``  
layers a ``Label`` on top of the color  
/// ``ShapeStyle/teal``. The background  
modifier inserts the  
/// regular material below the label,  
blurring the part of
```

```
/// the background that the label ---  
including its padding --- covers:  
///  
/// ! [A screenshot of a label on a teal  
background, where the area behind  
/// the label appears blurred.]  
(Material-1)  
///  
/// A material isn't a view, but adding a  
material is like inserting a  
/// translucent layer between the  
modified view and its background:  
///  
/// ! [An illustration that shows a  
background layer below a material layer,  
/// which in turn appears below a  
foreground layer.] (Material-2)  
///  
/// The blurring effect provided by the  
material isn't simple opacity. Instead,  
/// it uses a platform-specific blending  
that produces an effect that resembles  
/// heavily frosted glass. You can see  
this more easily with a complex  
/// background, like an image:  
///  
///     ZStack {  
///         Image("chili_peppers")  
///             .resizable()  
///             .aspectRatio(contentMode:  
.fit)  
///         Label("Flag", systemImage:  
"flag.fill")
```

```
///     .padding()
///     .background(.regularMaterial)
/// }
///
/// ! [A screenshot of a label on an image
background, where the area behind
/// the label appears blurred.]  

(Material-3)
///
/// For physical materials, the degree to
which the background colors pass
/// through depends on the thickness. The
effect also varies with light and
/// dark appearance:
///
/// ! [An array of labels on a teal
background. The first column, labeled
light
/// appearance, shows a succession of
labels on blurred backgrounds where the
/// blur increases from top to bottom,
resulting in lighter and lighter blur.
/// The second column, labeled dark
appearance, shows a similar succession of
/// labels, except that the blur gets
darker from top to bottom. The rows are
/// labeled, from top to bottom: no
material, ultra thin, thin, regular,
thick,
/// and ultra thick.]  

(Material-4)
///
/// If you need a material to have a
```

particular shape, you can use the `View/background(_:in:fillStyle:)` modifier. For example, you can create a material with rounded corners:

```
///  
///     ZStack {  
///         Color.teal  
///         Label("Flag", systemImage:  
"flag.fill")  
///             .padding()  
///             .background(.regularMater  
ial, in: RoundedRectangle(cornerRadius:  
8))  
///     }  
///  
/// ! [A screenshot of a label on a teal  
background, where the area behind  
the label appears blurred. The  
blurred area has rounded corners.]  
(Material-5)  
///  
/// When you add a material, foreground  
elements exhibit vibrancy,  
/// a context-specific blend of the  
foreground and background colors  
/// that improves contrast. However using  
`View/foregroundStyle(_:)``  
/// to set a custom foreground style ---  
excluding the hierarchical styles,  
/// like `ShapeStyle/secondary-  
swift.type.property`` --- disables  
vibrancy.
```

```
///  
/// > Note: A material blurs a background  
/// that's part of your app, but not  
/// what appears behind your app on the  
/// screen.  
/// For example, the content on the Home  
/// Screen doesn't affect the appearance  
/// of a widget.  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
public struct Material : Sendable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 10.0, *)  
extension Material {  
  
    /// A material that's somewhat  
    translucent.  
    public static let regular: Material  
  
    /// A material that's more opaque  
    than translucent.  
    public static let thick: Material  
  
    /// A material that's more  
    translucent than opaque.  
    public static let thin: Material  
  
    /// A mostly translucent material.  
    public static let ultraThin: Material  
  
    /// A mostly opaque material.
```

```
    public static let ultraThick:  
Material  
}  
  
@available(iOS 15.0, macOS 12.0, *)  
@available(tvOS, unavailable)  
@available(watchOS, unavailable)  
extension Material {  
  
    /// A material matching the style of  
    system toolbars.  
    public static let bar: Material  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Material : ShapeStyle {  
  
    /// The type of shape style this will  
    resolve to.  
    ///  
    /// When you create a custom shape  
    style, Swift infers this type  
    /// from your implementation of the  
    required `resolve` function.  
    @available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
    public typealias Resolved = Never  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension Material {
```

```
    /// Sets an explicit active
    appearance for this material.
    ///
    /// Materials used as the `window`
    container background on macOS will
    /// automatically appear inactive
    when their the window appears inactive,
    but
    /// can be made to always appear
    active by setting the active appearance
    /// behavior to be always active:
    ///
    ///     Text("Hello, World!")
    ///         .containerBackground(
    ///
Material.regular.materialActiveAppearance
(.active),
    ///                         for: .window)
    ///
    public func
materialActiveAppearance(_ appearance:
MaterialActiveAppearance) -> Material
}

/// The behavior for how materials appear
active and inactive.
///
/// On macOS, materials have active and
inactive appearances that can reinforce
/// the active appearance of the window
they are in:
/// - Materials used as a `window`
```

```
container background and `bar` materials
/// will appear inactive when their
containing window is inactive.
/// - All other materials will always
appear active by default.
///
/// An explicit active appearance can be
set to override a material's default
/// behavior. For example, materials used
as the `window` container background
/// can be made to always appear active
by setting the active appearance
/// behavior to be always active:
///
///     Text("Hello, World!")
///         .containerBackground(
///
Material.regular.materialActiveAppearance
(.active),
///             for: .window)
///
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct MaterialActiveAppearance :  
Sendable, Equatable {  
  
    /// Materials will automatically
    appear active or inactive based on
    context
    /// and platform convention.
    ///
    /// Examples of automatic behavior on
macOS:
```

```
    /// - Materials used as a `window`  
    container background will appear inactive  
    /// when the containing window is  
    inactive.
```

```
    /// - `Material.bar` materials will  
    appear inactive when the containing  
    /// window is inactive.
```

```
    /// - All other materials will appear  
    as active.
```

```
    ///
```

```
    /// Materials always appear as active  
    on all other platforms.
```

```
public static let automatic:  
MaterialActiveAppearance
```

```
    /// Materials will always appear  
    active.
```

```
public static let active:  
MaterialActiveAppearance
```

```
    /// Materials will always appear  
    inactive.
```

```
@available(iOS, unavailable)  
@available(tvOS, unavailable)  
@available(watchOS, unavailable)  
@available(visionOS, unavailable)  
public static let inactive:  
MaterialActiveAppearance
```

```
    /// Materials will have an active or  
    inactive appearance based on
```

```
    /// the active appearance of their  
    window.
```

```
///  
/// On non-macOS platforms, materials  
will always appear as active.  
public static let matchWindow:  
MaterialActiveAppearance  
  
    /// 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:  
MaterialActiveAppearance, b:  
MaterialActiveAppearance) -> Bool  
}  
  
/// A two-dimensional gradient defined by  
a 2D grid of positioned  
/// colors.  
///  
/// Each vertex has a position, a color  
and four surrounding Bezier  
/// control points (leading, top,  
trailing, bottom) that define the  
/// tangents connecting the vertex with  
its four neighboring vertices.
```

```
/// (Vertices on the corners or edges of  
the mesh have less than four  
neighbors, they ignore their extra  
control points.) Control points  
/// may either be specified explicitly or  
implicitly.  
///  
/// When rendering, a tessellated  
sequence of Bezier patches are  
/// created, and vertex colors are  
interpolated across each patch,  
/// either linearly, or via another set  
of cubic curves derived from  
/// how the colors change between  
neighbors -- the latter typically  
/// gives smoother color transitions.  
///  
///     MeshGradient(width: 3, height: 3,  
points: [  
///         .init(0, 0), .init(0.5,  
0), .init(1, 0),  
///         .init(0, 0.5), .init(0.5,  
0.5), .init(1, 0.5),  
///         .init(0, 1), .init(0.5,  
1), .init(1, 1)  
///     ], colors: [  
///         .red, .purple, .indigo,  
///         .orange, .white, .blue,  
///         .yellow, .green, .mint  
///     ])  
///  
@available(iOS 18.0, macOS 15.0, watchOS  
11.0, tvOS 18.0, visionOS 2.0, *)
```

```
public struct MeshGradient : ShapeStyle,  
Equatable, Sendable {  
  
    /// An array of 2D locations and  
    their control points.  
    public enum Locations : Equatable,  
Sendable {  
  
        /// Vertices are only specified  
        as their location, their control  
        /// points are inferred from the  
        locations of their neighbors.  
        case points([SIMD2<Float>])  
  
        /// Vertices explicitly  
        specifying their location and control  
        /// points.  
        case  
bezierPoints([MeshGradient.BezierPoint])  
  
        /// 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:
```

```
MeshGradient.Locations, b:  
MeshGradient.Locations) -> Bool  
}  
  
    /// An array of colors.  
    public enum Colors : Equatable,  
Sendable {  
  
        case colors([Color])  
  
        case  
resolvedColors([Color.Resolved])  
  
            /// 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:  
MeshGradient.Colors, b:  
MeshGradient.Colors) -> Bool  
    }  
  
    /// One location in a gradient mesh,  
along with the four Bezier  
    /// control points surrounding it.
```

```
    @frozen public struct BezierPoint :  
Equatable, Sendable {  
  
    /// The position of the vertex.  
    public var position: SIMD2<Float>  
  
    /// The Bezier control point of  
the vertex's leading edge.  
    public var leadingControlPoint:  
SIMD2<Float>  
  
    /// The Bezier control point of  
the vertex's top edge.  
    public var topControlPoint:  
SIMD2<Float>  
  
    /// The Bezier control point of  
the vertex's trailing edge.  
    public var trailingControlPoint:  
SIMD2<Float>  
  
    /// The Bezier control point of  
the vertex's bottom edge.  
    public var bottomControlPoint:  
SIMD2<Float>  
  
    /// Creates a new vertex.  
    ///  
    /// - Parameters:  
    ///     - position: the position of  
the vertex in the coordinate  
    ///     space the gradient is  
interpreted in.
```

```
    /// - leadingControlPoint: the
Bezier control point of the
    /// vertex's leading edge.
    /// - topControlPoint: the
Bezier control point of the
    /// vertex's top edge.
    /// - trailingControlPoint: the
Bezier control point of the
    /// vertex's trailing edge.
    /// - bottomControlPoint: the
Bezier control point of the
    /// vertex's bottom edge.
public init(position:
SIMD2<Float>, leadingControlPoint:
SIMD2<Float>, topControlPoint:
SIMD2<Float>, trailingControlPoint:
SIMD2<Float>, bottomControlPoint:
SIMD2<Float>)
```

```
    /// 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:
MeshGradient.BezierPoint, b:
```

```
MeshGradient.BezierPoint) -> Bool
}

    /// The width of the mesh, i.e. the
    number of vertices per row.
    public var width: Int

    /// The height of the mesh, i.e. the
    number of vertices per column.
    public var height: Int

    /// The array of locations. Must
    contain `width x height` elements.
    public var locations:
    MeshGradient.Locations

    /// The array of colors. Must contain
    `width x height` elements.
    public var colors:
    MeshGradient.Colors

    /// The background color, this fills
    any points outside the defined
    /// vertex mesh.
    public var background: Color

    /// Whether cubic (smooth)
    interpolation should be used for the
    /// colors in the mesh (rather than
    only for the shape of the
    /// mesh).
    public var smoothsColors: Bool
```

```
    /// The color space in which to
    /// interpolate vertex colors.
    public var colorSpace:
Gradient.ColorSpace

    /// Creates a new gradient mesh
    /// specified as a 2D grid of colored
    /// vertices.
    ///
    /// - Parameters:
    ///   - width: the width of the mesh,
    ///     i.e. the number of vertices
    ///     per row.
    ///   - height: the height of the
    ///     mesh, i.e. the number of vertices
    ///     per column.
    ///   - locations: the array of
    ///     locations, containing `width x
    ///     height` elements.
    ///   - colors: the array of colors,
    ///     containing `width x height`
    ///     elements.
    ///   - background: the background
    ///     color, this fills any points
    ///     outside the defined vertex
    ///     mesh.
    ///   - smoothsColors: whether cubic
    ///     (smooth) interpolation should
    ///     be used for the colors in the
    ///     mesh (rather than only for the
    ///     shape of the mesh).
    ///   - colorSpace: the color space
    ///     in which to interpolate vertex
```

```
    /// colors.  
    public init(width: Int, height: Int,  
locations: MeshGradient.Locations,  
colors: MeshGradient.Colors, background:  
Color = .clear, smoothsColors: Bool =  
true, colorSpace: Gradient.ColorSpace  
= .device)  
  
        /// Creates a new gradient mesh  
specified as a 2D grid of colored  
        /// points.  
        ///  
        /// - Parameters:  
        ///     - width: the width of the mesh,  
i.e. the number of vertices  
        ///         per row.  
        ///     - height: the height of the  
mesh, i.e. the number of vertices  
        ///         per column.  
        ///     - points: the array of points,  
containing `width x height`  
        ///         elements.  
        ///     - colors: the array of colors,  
containing `width x height`  
        ///         elements.  
        ///     - background: the background  
color, this fills any points  
        ///         outside the defined vertex  
mesh.  
        ///     - smoothsColors: whether cubic  
(smooth) interpolation should  
        ///         be used for the colors in the  
mesh (rather than only for the
```

```
    ///      shape of the mesh).
    ///      - colorSpace: the color space
in which to interpolate vertex
    ///      colors.
public init(width: Int, height: Int,
points: [SIMD2<Float>], colors: [Color],
background: Color = .clear,
smoothsColors: Bool = true, colorSpace:
Gradient.ColorSpace = .device)

    /// Creates a new gradient mesh
specified as a 2D grid of colored
    /// points, with already-resolved
SRGB colors.
    ///
    /// - Parameters:
    ///      - width: the width of the mesh,
i.e. the number of vertices
    ///      per row.
    ///      - height: the height of the
mesh, i.e. the number of vertices
    ///      per column.
    ///      - points: the array of points,
containing `width x height`
    ///      elements.
    ///      - resolvedColors: the array of
colors, containing `width x
    ///      height` elements.
    ///      - background: the background
color, this fills any points
    ///      outside the defined vertex
mesh.
    ///      - smoothsColors: whether cubic
```

```
(smooth) interpolation should
    ///      be used for the colors in the
mesh (rather than only for the
    ///      shape of the mesh).
    /// - colorSpace: the color space
in which to interpolate vertex
    ///      colors.

public init(width: Int, height: Int,
points: [SIMD2<Float>], resolvedColors:
[Color.Resolved], background: Color
= .clear, smoothsColors: Bool = true,
colorSpace: Gradient.ColorSpace
= .device)

    /// Creates a new gradient mesh
specified as a 2D grid of colored
    /// points, specifying the Bezier
control points explicitly.
    ///
    /// - Parameters:
    ///   - width: the width of the mesh,
i.e. the number of vertices
    ///      per row.
    ///   - height: the height of the
mesh, i.e. the number of vertices
    ///      per column.
    ///   - bezierPoints: the array of
points and control points,
    ///      containing `width x height`
elements.
    ///   - colors: the array of colors,
containing `width x height`
    ///      elements.
```

```
    /// - background: the background
color, this fills any points
    /// outside the defined vertex
mesh.
    /// - smoothsColors: whether cubic
(smooth) interpolation should
    /// be used for the colors in the
mesh (rather than only for the
    /// shape of the mesh).
    /// - colorSpace: the color space
in which to interpolate vertex
    /// colors.
public init(width: Int, height: Int,
bezierPoints: [MeshGradient.BezierPoint],
colors: [Color], background: Color
= .clear, smoothsColors: Bool = true,
colorSpace: Gradient.ColorSpace
= .device)

    /// Creates a new gradient mesh
specified as a 2D grid of colored
    /// points, specifying the Bezier
control points explicitly, with
    /// already-resolved sRGB colors.
    ///
    /// - Parameters:
    ///   - width: the width of the mesh,
i.e. the number of vertices
    ///   per row.
    ///   - height: the height of the
mesh, i.e. the number of vertices
    ///   per column.
    ///   - bezierPoints: the array of
```

```
points and control points,  
    /// containing `width x height`  
elements.  
    /// - resolvedColors: the array of  
colors, containing `width x  
    /// height` elements.  
    /// - background: the background  
color, this fills any points  
    /// outside the defined vertex  
mesh.  
    /// - smoothsColors: whether cubic  
(smooth) interpolation should  
    /// be used for the colors in the  
mesh (rather than only for the  
    /// shape of the mesh).  
    /// - colorSpace: the color space  
in which to interpolate vertex  
    /// colors.  
public init(width: Int, height: Int,  
bezierPoints: [MeshGradient.BezierPoint],  
resolvedColors: [Color.Resolved],  
background: Color = .clear,  
smoothsColors: Bool = true, colorSpace:  
Gradient.ColorSpace = .device)  
  
    /// 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:  
        MeshGradient, b: MeshGradient) -> Bool  
  
    /// The type of view representing the  
    body of this view.  
    ///  
    /// When you create a custom view,  
    Swift infers this type from your  
    /// implementation of the required  
    ``View/body-swift.property`` property.  
    @available(iOS 18.0, tvOS 18.0,  
    watchOS 11.0, visionOS 2.0, macOS 15.0,  
    *)  
    public typealias Body  
  
    /// The type of shape style this will  
    resolve to.  
    ///  
    /// When you create a custom shape  
    style, Swift infers this type  
    /// from your implementation of the  
    required `resolve` function.  
    @available(iOS 18.0, tvOS 18.0,  
    watchOS 11.0, visionOS 2.0, macOS 15.0,  
    *)  
    public typealias Resolved = Never  
}  
  
@available(iOS 18.0, macOS 15.0, watchOS
```

```
11.0, tvOS 18.0, visionOS 2.0, *)
extension MeshGradient : View {
}

@available(iOS 18.0, macOS 15.0, watchOS
11.0, tvOS 18.0, visionOS 2.0, *)
extension MeshGradient.BezierPoint :
BitwiseCopyable {
}

/// A value with a modifier applied to
it.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct
ModifiedContent<Content, Modifier> {

    /// The type of view representing the
body of this view.
    ///
    /// When you create a custom view,
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
    public typealias Body = Never

    /// The content that the modifier
transforms into a new view or new
    /// view modifier.
    public var content: Content

    /// The view modifier.
    public var modifier: Modifier
```

```
    /// A structure that the defines the
    content and modifier needed to produce
    /// a new view or view modifier.
    ///
    /// If `content` is a ``View`` and
    `modifier` is a ``ViewModifier``, the
    /// result is a ``View``. If
    `content` and `modifier` are both view
    /// modifiers, then the result is a
    new ``ViewModifier`` combining them.
    ///
    /// - Parameters:
    ///   - content: The content that
    the modifier changes.
    ///   - modifier: The modifier to
    apply to the content.
    @inlinable nonisolated public
    init(content: Content, modifier:
    Modifier)
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ModifiedContent : Equatable
where Content : Equatable, Modifier :
Equatable {

    /// 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:
ModifiedContent<Content, Modifier>, b:
ModifiedContent<Content, Modifier>) ->
Bool
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ModifiedContent : View where
Content : View, Modifier : ViewModifier {

    /// The content and behavior of the
view.
    /**
     * When you implement a custom view,
you must implement a computed
     * `body` property to provide the
content for your view. Return a view
     * that's composed of built-in views
that SwiftUI provides, plus other
     * composite views that you've
already defined:
    /**
     * struct MyView: View {
     *     var body: some View {
     *         Text("Hello, World!")

```

```
    /**
     */
    /**
     * For more information about
     * composing views and a view hierarchy,
     * see <doc:Declaring-a-Custom-
     * View>.
     */
    @MainActor @preconcurrency public var
    body: ModifiedContent<Content,
    Modifier>.Body { get }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ModifiedContent : ViewModifier
where Content : ViewModifier, Modifier :
ViewModifier {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ModifiedContent :
DynamicViewContent where Content :
DynamicViewContent, Modifier :
ViewModifier {

    /// The collection of underlying
    data.
    public var data: Content.Data { get }

    /// The type of the underlying
    collection of data.
    @available(iOS 13.0, tvOS 13.0,
```

```
watchOS 6.0, macOS 10.15, *)
    public typealias Data = Content.Data
}

/// A keyframe that immediately moves to
/// the given value without interpolating.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct MoveKeyframe<Value> :
KeyframeTrackContent where Value : Animatable {

    /// Creates a new keyframe using the
    /// given value.
    ///
    /// - Parameters:
    ///   - to: The value of the
    ///     keyframe.
    public init(_ to: Value)

    @available(iOS 17.0, tvOS 17.0,
    watchOS 10.0, macOS 14.0, *)
        public typealias Body =
MoveKeyframe<Value>
}

/// Returns a transition that moves the
/// view away, towards the specified
/// edge of the view.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@MainActor @preconcurrency public struct
MoveTransition : Transition {
```

```
    /// The edge to move the view
    towards.
    @MainActor @preconcurrency public var
    edge: Edge

        /// Creates a transition that moves
    the view away, towards the specified
        /// edge of the view.
    @MainActor @preconcurrency public
    init(edge: Edge)

        /// Gets the current body of the
    caller.
        ///
        /// `content` is a proxy for the view
    that will have the modifier
        /// represented by `Self` applied to
    it.
    @MainActor @preconcurrency public
    func body(content:
    MoveTransition.Content, phase:
    TransitionPhase) -> some View

        /// The type of view representing the
    body.
        @available(iOS 17.0, tvOS 17.0,
    watchOS 10.0, macOS 14.0, *)
        public typealias Body = some View
}

/// A named coordinate space.
```

```
///  
/// Use the `coordinateSpace(_:)`  
/// modifier to assign a name to the local  
/// coordinate space of a parent view.  
Child views can then refer to that  
/// coordinate space using ` `.named(_:)` .  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
public struct NamedCoordinateSpace :  
CoordinateSpaceProtocol, Equatable {  
  
    /// The resolved coordinate space.  
    public var coordinateSpace:  
CoordinateSpace { get }  
  
    /// 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:  
NamedCoordinateSpace, b:  
NamedCoordinateSpace) -> Bool  
}  
  
/// A dynamic property type that allows
```

```
access to a namespace defined
/// by the persistent identity of the
object containing the property
/// (e.g. a view).
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
@frozen @propertyWrapper public struct
Namespace : DynamicProperty, Sendable {

    @inlinable public init()

        public var wrappedValue: Namespace.ID
    { get }

        /// A namespace defined by the
        persistent identity of an
        /// `@Namespace` dynamic property.
        @frozen public struct ID : Hashable {

            /// 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: Namespace.ID, b: Namespace.ID) -> 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 }
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension Namespace : BitwiseCopyable {
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension Namespace.ID : Sendable {
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension Namespace.ID : BitwiseCopyable
```

```
{  
}
```

```
/// A property wrapper type that  
subscribes to an observable object and  
/// invalidates a view whenever the  
observable object changes.  
///  
/// Add the `@ObservedObject` attribute  
to a parameter of a SwiftUI ``View``  
/// when the input is an  
///  
<doc://com.apple.documentation/documentation/Combine/ObservableObject>  
/// and you want the view to update when  
the object's published properties  
/// change. You typically do this to pass  
a ``StateObject`` into a subview.  
///  
/// The following example defines a data  
model as an observable object,  
/// instantiates the model in a view as a  
state object, and then passes  
/// the instance to a subview as an  
observed object:  
///  
///     class DataModel: ObservableObject  
{  
///         @Published var name = "Some  
Name"  
///         @Published var isEnabled =  
false  
///     }
```

```
///  
///     struct MyView: View {  
///         @StateObject private var  
model = DataModel()  
///  
///         var body: some View {  
///             Text(model.name)  
///             MySubView(model: model)  
///         }  
///     }  
///  
///     struct MySubView: View {  
///         @ObservedObject var model:  
DataModel  
///  
///         var body: some View {  
///             Toggle("Enabled", isOn:  
$model.isEnabled)  
///         }  
///     }  
///  
/// When any published property of the  
observable object changes, SwiftUI  
/// updates any view that depends on the  
object. Subviews can  
/// also make updates to the model  
properties, like the ``Toggle`` in the  
/// above example, that propagate to  
other observers throughout the view  
/// hierarchy.  
///  
/// Don't specify a default or initial  
value for the observed object. Use the
```

```
/// attribute only for a property that
acts as an input for a view, as in the
/// above example.
///
/// > Note: Don't wrap objects conforming
to the
///
<doc://com.apple.documentation/documentation/Observation/Observable>
/// protocol with `@ObservedObject`.
SwiftUI automatically tracks dependencies
/// to `Observable` objects used within
body and updates dependent views when
/// their data changes. Attempting to
wrap an `Observable` object with
/// `@ObservedObject` may cause a
compiler error, because it requires that
its
/// wrapped object to conform to the
///
<doc://com.apple.documentation/documentation/Combine/ObservableObject>
/// protocol.
///
>
/// > If the view needs a binding to a
property of an `Observable` object in
/// its body, wrap the object with the
``Bindable`` property wrapper instead;
/// for example, `@Bindable var model:
DataModel`. For more information, see
/// <doc:Managing-model-data-in-your-
app>.
```

**@available(iOS 13.0, macOS 10.15, tvOS**

```
13.0, watchOS 6.0, *)
@MainActor @propertyWrapper
@preconcurrency @frozen public struct
ObservedObject<ObjectType> :
DynamicProperty where ObjectType :
ObservableObject {

    /// A wrapper of the underlying
    observable object that can create
    bindings
    /// to its properties.
    @MainActor @dynamicMemberLookup
    @preconcurrency @frozen public struct
    Wrapper {

        /// Gets a binding to the value
        of a specified key path.
        ///
        /// - Parameter keyPath: A key
        path to a specific value.
        ///
        /// - Returns: A new binding.
        @MainActor @preconcurrency public
        subscript<Subject>(dynamicMember keyPath:
        ReferenceWritableKeyPath<ObjectType,
        Subject>) -> Binding<Subject> { get }
    }

    /// Creates an observed object with
    an initial value.
    ///
    /// This initializer has the same
    behavior as the ``init(wrappedValue:)``
```

```
    /// initializer. See that initializer  
for more information.  
    ///  
    /// - Parameter initialValue: An  
initial value.  
    @MainActor @preconcurrency public  
init(initialValue: ObjectType)  
  
    /// Creates an observed object with  
an initial wrapped value.  
    ///  
    /// Don't call this initializer  
directly. Instead, declare  
    /// an input to a view with the  
`@ObservedObject` attribute, and pass a  
    /// value to this input when you  
instantiate the view. Unlike a  
    /// ``StateObject`` which manages  
data storage, you use an observed  
    /// object to refer to storage that  
you manage elsewhere, as in the  
    /// following example:  
    ///  
    ///     class DataModel:  
ObservableObject {  
    ///             @Published var name =  
"Some Name"  
    ///             @Published var isEnabled  
= false  
    ///         }  
    ///  
    ///     struct MyView: View {  
    ///         @StateObject private var
```



```
    /// The underlying value that the
    observed object references.
    ///
    /// The wrapped value property
    provides primary access to the observed
    /// object's data. However, you don't
    typically access it by name. Instead,
    /// SwiftUI accesses this property
    for you when you refer to the variable
    /// that you create with the
`@ObservedObject` attribute.
    ///
    /// struct MySubView: View {
    ///     @ObservedObject var
model: DataModel
    ///
    ///         var body: some View {
    ///             Text(model.name) //
Reads name from model's wrapped value.
    ///         }
    ///     }
    ///
    /// When you change a wrapped value,
you can access the new value
    /// immediately. However, SwiftUI
updates views that display the value
    /// asynchronously, so the interface
might not update immediately.
    @MainActor @preconcurrency public var
wrappedValue: ObjectType

    /// A projection of the observed
object that creates bindings to its
```

```
    /// properties.  
    ///  
    /// Use the projected value to get a  
``Binding`` to a property of an  
    /// observed object. To access the  
projected value, prefix the property  
    /// variable with a dollar sign  
(`$`). For example, you can get a binding  
    /// to a model's `isEnabled` Boolean  
so that a ``Toggle`` can control its  
    /// value:  
    ///  
    ///     struct MySubView: View {  
    ///         @ObservedObject var  
model: DataModel  
    ///  
    ///             var body: some View {  
    ///                 Toggle("Enabled",  
isOn: $model.isEnabled)  
    ///             }  
    ///         }  
    ///  
    /// > Important: A `Binding` created  
by the projected value must only be  
    /// read from, or written to by the  
main actor. Failing to do so may result  
    /// in undefined behavior, or data  
loss. When this occurs, SwiftUI will  
    /// issue a runtime warning. In a  
future release, a crash will occur  
    /// instead.  
    @MainActor @preconcurrency public var  
projectedValue:
```

```
ObservedObject<ObjectType>.Wrapper {  
    get }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ObservedObject : Sendable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ObservedObject.Wrapper :  
Sendable {  
}  
  
/// A shape with a translation offset  
transform applied to it.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct  
OffsetShape<Content> : Shape where  
Content : Shape {  
  
    public var shape: Content  
  
    public var offset: CGSize  
  
    @inlinable public init(shape:  
Content, offset: CGSize)  
  
        /// Describes this shape as a path  
within a rectangular frame of reference.  
        ///
```

```
    /// - Parameter rect: The frame of  
    reference for describing this shape.  
    ///  
    /// - Returns: A path that describes  
    this shape.  
    nonisolated public func path(in rect:  
    CGRect) -> Path
```

```
    /// An indication of how to style a  
    shape.  
    ///  
    /// SwiftUI looks at a shape's role  
    when deciding how to apply a  
    /// ``ShapeStyle`` at render time.  
    The ``Shape`` protocol provides a  
    /// default implementation with a  
    value of ``ShapeRole/fill``. If you  
    /// create a composite shape, you can  
    provide an override of this property  
    /// to return another value, if  
    appropriate.  
    @available(iOS 15.0, macOS 12.0, tvOS  
    15.0, watchOS 8.0, *)  
    nonisolated public static var role:  
    ShapeRole { get }  
  
    /// Returns the behavior this shape  
    should use for different layout  
    /// directions.  
    ///  
    /// If the layoutDirectionBehavior  
    for a Shape is one that mirrors, the  
    /// shape's path will be mirrored
```

```
horizontally when in the specified layout
    /// direction. When mirrored, the
    individual points of the path will be
    /// transformed.
    ///
    /// Defaults to ` `.mirrors` when
    deploying on iOS 17.0, macOS 14.0,
    /// tvOS 17.0, watchOS 10.0 and
    later, and to ` `.fixed` if not.
    /// To mirror a path when deploying
    to earlier releases, either use
    ///
`View.flipsForRightToLeftLayoutDirection`
for a filled or stroked
    /// shape or conditionally mirror the
    points in the path of the shape.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    nonisolated public var
layoutDirectionBehavior:
LayoutDirectionBehavior { get }

    /// The type defining the data to
animate.
    public typealias AnimatableData =
AnimatablePair<Content.AnimatableData,
CGSize.AnimatableData>

    /// The data to animate.
    public var animatableData:
OffsetShape<Content>.AnimatableData

    /// The type of view representing the
```

body of this view.

```
    /**
     * When you create a custom view,
     Swift infers this type from your
     implementation of the required
     ``View/body-swift.property`` property.
     @available(iOS 13.0, tvOS 13.0,
     watchOS 6.0, macOS 10.15, *)
     public typealias Body
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension OffsetShape : InsettableShape
where Content : InsettableShape {

    /// Returns `self` inset by `amount`.
    @inlinable nonisolated public func
    inset(by amount: CGFloat) ->
    OffsetShape<Content.InsetShape>

    /// The type of the inset shape.
    @available(iOS 13.0, tvOS 13.0,
    watchOS 6.0, macOS 10.15, *)
    public typealias InsetShape =
    OffsetShape<Content.InsetShape>
}

/// Returns a transition that offset the
view by the specified amount.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@MainActor @preconcurrency public struct
```

```
OffsetTransition : Transition {  
  
    /// The amount to offset the view by.  
    @MainActor @preconcurrency public var  
    offset: CGSize  
  
    /// Creates a transition that offset  
    the view by the specified amount.  
    @MainActor @preconcurrency public  
    init(_ offset: CGSize)  
  
    /// Gets the current body of the  
    caller.  
    ///  
    /// `content` is a proxy for the view  
    that will have the modifier  
    /// represented by `Self` applied to  
    it.  
    @MainActor @preconcurrency public  
    func body(content:  
OffsetTransition.Content, phase:  
TransitionPhase) -> some View  
  
    /// The type of view representing the  
    body.  
    @available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
    public typealias Body = some View  
}  
  
/// A transition from transparent to  
opaque on insertion, and from opaque to
```

```
/// transparent on removal.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
@MainActor @preconcurrency public struct  
OpacityTransition : Transition {  
  
    @MainActor @preconcurrency public  
    init()  
  
        /// Gets the current body of the  
        caller.  
        ///  
        /// `content` is a proxy for the view  
        that will have the modifier  
        /// represented by `Self` applied to  
        it.  
        @MainActor @preconcurrency public  
        func body(content:  
        OpacityTransition.Content, phase:  
        TransitionPhase) -> some View  
  
        /// Returns the properties this  
        transition type has.  
        ///  
        /// Defaults to  
        `TransitionProperties()`.  
        @MainActor @preconcurrency public  
        static let properties:  
        TransitionProperties  
  
        /// The type of view representing the  
        body.
```

```
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
public typealias Body = some View
}

/// An action that opens a URL.
///
/// Read the
``EnvironmentValues/openURL`` environment
value to get an
/// instance of this structure for a
given ``Environment``. Call the
/// instance to open a URL. You call the
instance directly because it
/// defines a
``OpenURLAction/callAsFunction(_:)``
method that Swift
/// calls when you call the instance.
///
/// For example, you can open a web site
when the user taps a button:
///
///     struct OpenURLExample: View {
///         @Environment(\.openURL)
private var openURL
///
///             var body: some View {
///                 Button {
///                     if let url =
URL(string: "https://www.example.com") {
///                         openURL(url)
///                     }
///                 } label: {
```

```
///                                Label("Get Help",
systemImage: "person.fill.questionmark")
///                                }
///                                }
///                                }
///                                }

/// If you want to know whether the
action succeeds, add a completion
/// handler that takes a Boolean value.
In this case, Swift implicitly
/// calls the
``OpenURLAction/callAsFunction(_:completi
on:)`` method
/// instead. That method calls your
completion handler after it determines
/// whether it can open the URL, but
possibly before it finishes opening
/// the URL. You can add a handler to the
example above so that
/// it prints the outcome to the console:
///
///     openURL(url) { accepted in
///         print(accepted ? "Success" :
"Failure")
///     }
///

/// The system provides a default open
URL action with behavior
/// that depends on the contents of the
URL. For example, the default
/// action opens a Universal Link in the
associated app if possible,
/// or in the user's default web browser
```

```
if not.  
///  
/// You can also set a custom action  
using the ``View/environment(_:_:)``  
/// view modifier. Any views that read  
the action from the environment,  
/// including the built-in ``Link`` view  
and ``Text`` views with markdown  
/// links, or links in attributed  
strings, use your action. Initialize an  
/// action by calling the  
``OpenURLAction/init(handler:)``  
initializer with  
/// a handler that takes a URL and  
returns an ``OpenURLAction/Result``:  
///  
///     Text("Visit [Example Company]  
(https://www.example.com) for details."  
///         .environment(\.openURL,  
OpenURLAction { url in  
///             handleURL(url) // Define  
this method to take appropriate action.  
///             return .handled  
///         })  
///  
/// SwiftUI translates the value that  
your custom action's handler  
/// returns into an appropriate Boolean  
result for the action call.  
/// For example, a view that uses the  
action declared above  
/// receives `true` when calling the  
action, because the
```

```
/// handler always returns
``OpenURLAction/Result/handled``.
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
@MainActor @preconcurrency public struct
OpenURLAction {

    /// The result of a custom open URL
    /// action.
    ///
    /// If you declare a custom
    ``OpenURLAction`` in the ``Environment``,
    /// return one of the result values
    from its handler.
    ///
    /// * Use ``handled`` to indicate
    that the handler opened the URL.
    /// * Use ``discarded`` to indicate
    that the handler discarded the URL.
    /// * Use ``systemAction`` without an
    argument to ask SwiftUI
    /// to open the URL with the system
    handler.
    /// * Use ``systemAction(_:)`` with a
    URL argument to ask SwiftUI
    /// to open the specified URL with
    the system handler.
    ///
    /// You can use the last option to
    transform URLs, while
    /// still relying on the system to
    open the URL. For example,
    /// you could append a path component
```

to every URL:

```
///  
///     .environment(\.openURL,  
OpenURLAction { url in  
    ///         .systemAction(url.appendi  
ngPathComponent("edit"))  
    ///     })  
    ///  
    /// @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
public struct Result : Sendable {  
  
    /// The handler opened the URL.  
    ///  
    /// The action invokes its  
completion handler with `true` when your  
    /// handler returns this value.  
    public static let handled:  
OpenURLAction.Result  
  
    /// The handler discarded the  
URL.  
    ///  
    /// The action invokes its  
completion handler with `false` when your  
    /// handler returns this value.  
    public static let discarded:  
OpenURLAction.Result  
  
    /// The handler asks the system  
to open the original URL.  
    ///  
    /// The action invokes its
```

```
completion handler with a value that
    /// depends on the outcome of the
system's attempt to open the URL.
    public static let systemAction:
OpenURLAction.Result

        /// The handler asks the system
to open the modified URL.
        ///
        /// The action invokes its
completion handler with a value that
        /// depends on the outcome of the
system's attempt to open the URL.
        ///
        /// - Parameter url: The URL that
the handler asks the system to open.
    public static func systemAction(_
url: URL) -> OpenURLAction.Result
}

/// Creates an action that opens a
URL.
///
/// Use this initializer to create a
custom action for opening URLs.
/// Provide a handler that takes a
URL and returns an
    /// ``OpenURLAction/Result``. Place
your handler in the environment
    /// using the
``View/environment(_:_:)`` view modifier:
    ///
    ///     Text("Visit [Example Company]
```

```
(https://www.example.com) for details.")

    /// .environment(\.openURL,
OpenURLAction { url in
    /// handleURL(url) //
Define this method to take appropriate
action.
    /// return .handled
    /// })
    ///
    /// Any views that read the action
from the environment, including the
    /// built-in ``Link`` view and
``Text`` views with markdown links, or
    /// links in attributed strings, use
your action.
    ///
    /// SwiftUI translates the value that
your custom action's handler
    /// returns into an appropriate
Boolean result for the action call.
    /// For example, a view that uses the
action declared above
    /// receives `true` when calling the
action, because the
    /// handler always returns
``OpenURLAction/Result/handled``.
    ///
    /// - Parameter handler: The closure
to run for the given URL.
    /// The closure takes a URL as
input, and returns a ``Result``
    /// that indicates the outcome of
the action.
```

```
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    @MainActor @preconcurrency public  
init(handler: @escaping (URL) ->  
OpenURLAction.Result)  
  
    /// Opens a URL, following system  
conventions.  
    ///  
    /// Don't call this method directly.  
SwiftUI calls it when you  
    /// call the ``OpenURLAction``  
structure that you get from the  
    /// ``Environment``, using a URL as  
an argument:  
    ///  
    /// struct OpenURLExample: View {  
    ///     @Environment(\.openURL)  
private var openURL  
    ///  
    ///         var body: some View {  
    ///             Button {  
    ///                 if let url =  
URL(string: "https://www.example.com") {  
    ///  
openURL(url) // Implicitly calls  
openURL.callAsFunction(url)  
    ///             }  
    ///         } label: {  
    ///             Label("Get Help",  
systemImage: "person.fill.questionmark")  
    ///         }  
    ///     }
```

```
    /**
     */
    /**
     * For information about how Swift
     * uses the `callAsFunction()` method to
     * simplify call site syntax, see
     * [Methods with Special Names]
     (https://docs.swift.org/swift-book/ReferenceManual/Declarations.html#ID622)
     * in *The Swift Programming
     Language*.
    /**
     * - Parameter url: The URL to open.
     @MainActor @preconcurrency public
func callAsFunction(_ url: URL)

    /**
     * Asynchronously opens a URL,
     following system conventions.
    /**
     * Don't call this method directly.
SwiftUI calls it when you
    /**
     * call the ``OpenURLAction``
     structure that you get from the
    /**
     * ``Environment``, using a URL and
     a completion handler as arguments:
    /**
    /**
     struct OpenURLExample: View {
    /**
         @Environment(\.openURL)
private var openURL
    /**
    /**
         var body: some View {
    /**
             Button {
    /**
                     if let url =
URL(string: "https://www.example.com") {
```

```
    ///                                     // Implicitly
calls openURL.callAsFunction(url) { ... }
    ///                                     openURL(url)
{ accepted in
    ///
print(accepted ? "Success" : "Failure")
    ///
}
    ///
} label: {
    ///
Label("Get Help",
systemImage: "person.fill.questionmark")
    ///
}
    ///
}
    ///
}
    ///
/// For information about how Swift
uses the `callAsFunction()` method to
    ///
// simplify call site syntax, see
    ///
// [Methods with Special Names]
(https://docs.swift.org/swift-book/ReferenceManual/Declarations.html#ID622)
    ///
in *The Swift Programming Language*.
    ///
    ///
- Parameters:
    ///
- url: The URL to open.
    ///
- completion: A closure the
method calls after determining if
    ///
it can open the URL, but
possibly before fully opening the URL.
    ///
The closure takes a Boolean
value that indicates whether the
    ///
method can open the URL.
```

```
@available(watchOS, unavailable)
@MainActor @preconcurrency public
func callAsFunction(_ url: URL,
completion: @escaping (_ accepted: Bool)
-> Void)
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension OpenURLAction : Sendable {
}

/// The outline of a 2D shape.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct Path : Equatable,
LosslessStringConvertible, @unchecked
Sendable {

    /// Creates an empty path.
    public init()

    /// Creates a path from an immutable
shape path.
    ///
    /// - Parameter path: The immutable
CoreGraphics path to initialize
    /// the new path from.
    ///
    public init(_ path: CGPath)

    /// Creates a path from a copy of a
mutable shape path.
```

```
///  
/// - Parameter path: The  
CoreGraphics path to initialize the new  
/// path from.  
///  
public init(_ path: CGMutablePath)  
  
/// Creates a path containing a  
rectangle.  
///  
/// This is a convenience function  
that creates a path of a  
/// rectangle. Using this convenience  
function is more efficient  
/// than creating a path and adding a  
rectangle to it.  
///  
/// Calling this function is  
equivalent to using `minX` and related  
/// properties to find the corners of  
the rectangle, then using the  
/// `move(to:)`, `addLine(to:)`, and  
`closeSubpath()` functions to  
/// add the rectangle.  
///  
/// - Parameter rect: The rectangle  
to add.  
///  
public init(_ rect: CGRect)  
  
/// Creates a path containing a  
rounded rectangle.  
///
```

```
    /// This is a convenience function  
    /// that creates a path of a rounded  
    /// rectangle. Using this convenience  
    /// function is more efficient  
    /// than creating a path and adding a  
    /// rounded rectangle to it.  
    ///  
    /// - Parameters:  
    ///   - rect: A rectangle, specified  
    ///     in user space coordinates.  
    ///   - cornerSize: The size of the  
    ///     corners, specified in user space  
    ///     coordinates.  
    ///   - style: The corner style.  
    ///     Defaults to the `continuous` style  
    ///     if not specified.  
    ///  
    public init(roundedRect rect: CGRect,  
cornerSize: CGSize, style:  
RoundedCornerStyle = .continuous)
```

```
    /// Creates a path containing a  
    /// rounded rectangle.  
    ///  
    /// This is a convenience function  
    /// that creates a path of a rounded  
    /// rectangle. Using this convenience  
    /// function is more efficient  
    /// than creating a path and adding a  
    /// rounded rectangle to it.  
    ///  
    /// - Parameters:  
    ///   - rect: A rectangle, specified
```

```
in user space coordinates.  
    /// - cornerRadius: The radius of  
all corners of the rectangle,  
    /// specified in user space  
coordinates.  
    /// - style: The corner style.  
Defaults to the `continuous` style  
    /// if not specified.  
    ///  
    public init(roundedRect rect: CGRect,  
cornerRadius: CGFloat, style:  
RoundedCornerStyle = .continuous)  
  
    /// Creates a path as the given  
rounded rectangle, which may have  
    /// uneven corner radii.  
    ///  
    /// This is a convenience function  
that creates a path of a rounded  
    /// rectangle. Using this function is  
more efficient than creating  
    /// a path and adding a rounded  
rectangle to it.  
    ///  
    /// - Parameters:  
    /// - rect: A rectangle, specified  
in user space coordinates.  
    /// - cornerRadii: The radius of  
each corner of the rectangle,  
    /// specified in user space  
coordinates.  
    /// - style: The corner style.  
Defaults to the `continuous` style
```

```
    ///      if not specified.  
    ///  
    @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
    public init(roundedRect rect: CGRect,  
cornerRadii: RectangleCornerRadii, style:  
RoundedCornerStyle = .continuous)  
  
        /// Creates a path as an ellipse  
within the given rectangle.  
        ///  
        /// This is a convenience function  
that creates a path of an  
        /// ellipse. Using this convenience  
function is more efficient than  
        /// creating a path and adding an  
ellipse to it.  
        ///  
        /// The ellipse is approximated by a  
sequence of Bézier  
        /// curves. Its center is the  
midpoint of the rectangle defined by  
        /// the rect parameter. If the  
rectangle is square, then the  
        /// ellipse is circular with a radius  
equal to one-half the width  
        /// (or height) of the rectangle. If  
the rect parameter specifies a  
        /// rectangular shape, then the major  
and minor axes of the ellipse  
        /// are defined by the width and  
height of the rectangle.  
        ///
```

```
    /// The ellipse forms a complete
    /// subpath of the path—that
    /// is, the ellipse drawing starts
    /// with a move-to operation and
    /// ends with a close-subpath
    /// operation, with all moves oriented in
    /// the clockwise direction. If you
    /// supply an affine transform,
    /// then the constructed Bézier
    /// curves that define the
    /// ellipse are transformed before
    /// they are added to the path.
    ///
    /// – Parameter rect: The rectangle
    /// that bounds the ellipse.
    ///
    public init(ellipseIn rect: CGRect)

    /// Creates an empty path, then
    /// executes a closure to add its
    /// initial elements.
    ///
    /// – Parameter callback: The Swift
    /// function that will be called to
    /// initialize the new path.
    ///
    public init(_ callback: (inout Path)
-> ())

    /// Initializes from the result of a
    /// previous call to
    /// `Path.stringRepresentation`.
    /// Fails if the `string` does not
```

```
    /// describe a valid path.  
    public init?(_ string: String)  
  
        /// A description of the path that  
may be used to recreate the path  
        /// via `init?(:_)`.  
        public var description: String {  
get }  
  
        /// An immutable path representing  
the elements in the path.  
        public var cgPath: CGPath { get }  
  
        /// A Boolean value indicating  
whether the path contains zero elements.  
        public var isEmpty: Bool { get }  
  
        /// A rectangle containing all path  
segments.  
        ///  
        /// This is the smallest rectangle  
completely enclosing all points  
        /// in the path but not including  
control points for Bézier  
        /// curves.  
        public var boundingRect: CGRect { get }  
  
        /// Returns true if the path contains  
a specified point.  
        ///  
        /// If `eoFill` is true, this method  
uses the even-odd rule to define which
```

```
    /// points are inside the path.  
Otherwise, it uses the non-zero rule.  
    public func contains(_ p: CGPoint,  
eoFill: Bool = false) -> Bool  
  
    /// An element of a path.  
    @frozen public enum Element :  
Equatable {  
  
        /// A path element that  
terminates the current subpath (without  
closing  
        /// it) and defines a new current  
point.  
        case move(to: CGPoint)  
  
        /// A line from the previous  
current point to the given point, which  
        /// becomes the new current  
point.  
        case line(to: CGPoint)  
  
        /// A quadratic Bézier curve from  
the previous current point to the  
        /// given end-point, using the  
single control point to define the curve.  
        ///  
        /// The end-point of the curve  
becomes the new current point.  
        case quadCurve(to: CGPoint,  
control: CGPoint)  
  
        /// A cubic Bézier curve from the
```

```
previous current point to the given
    /// end-point, using the two
control points to define the curve.
    ///
    /// The end-point of the curve
becomes the new current point.
case curve(to: CGPoint, control1:
CGPoint, control2: CGPoint)

    /// A line from the start point
of the current subpath (if any) to the
    /// current point, which
terminates the subpath.
    ///
    /// After closing the subpath,
the current point becomes undefined.
case closeSubpath

    /// 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:
Path.Element, b: Path.Element) -> Bool
}
```

```
    /// Calls `body` with each element in  
the path.
```

```
    public func forEach(_ body:  
(Path.Element) -> Void)
```

```
    /// Returns a stroked copy of the  
path using `style` to define how the  
    /// stroked outline is created.
```

```
    public func strokedPath(_ style:  
StrokeStyle) -> Path
```

```
    /// Returns a partial copy of the  
path.
```

```
    ///  
    /// The returned path contains the  
region between `from` and `to`, both of  
    /// which must be fractions between  
zero and one defining points
```

```
    /// linearly-interpolated along the  
path.
```

```
    public func trimmedPath(from:  
CGFloat, to: CGFloat) -> Path
```

```
    /// 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: Path, b:
Path) -> Bool
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Path : Shape {

    /// Describes this shape as a path
within a rectangular frame of reference.
    ///
    /// - Parameter rect: The frame of
reference for describing this shape.
    ///
    /// - Returns: A path that describes
this shape.
    nonisolated public func path(in _:  
CGRect) -> Path

    /// The type defining the data to
animate.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias AnimatableData =
EmptyAnimatableData

    /// The type of view representing the
body of this view.
    ///
    /// When you create a custom view,
```

```
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
@available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
public typealias Body
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Path {

    /// Begins a new subpath at the
    specified point.
    ///
    /// The specified point becomes the
    start point of a new subpath.
    /// The current point is set to this
    start point.
    ///
    /// - Parameter end: The point, in
    user space coordinates, at which
    /// to start a new subpath.
    ///
    public mutating func move(to end:
    CGPoint)

    /// Appends a straight line segment
    from the current point to the
    /// specified point.
    ///
    /// After adding the line segment,
    the current point is set to the
```

```
    /// endpoint of the line segment.  
    ///  
    /// - Parameter end: The location, in  
    user space coordinates, for the  
    /// end of the new line segment.  
    ///  
    public mutating func addLine(to end:  
CGPoint)  
  
    /// Adds a quadratic Bézier curve to  
the path, with the  
    /// specified end point and control  
point.  
    ///  
    /// This method constructs a curve  
starting from the path's current  
    /// point and ending at the specified  
end point, with curvature  
    /// defined by the control point.  
After this method appends that  
    /// curve to the current path, the  
end point of the curve becomes  
    /// the path's current point.  
    ///  
    /// - Parameters:  
    ///      the curve.  
    /// - control: The control point of  
the curve, in user space  
    ///      coordinates.  
    ///  
    public mutating func addQuadCurve(to  
end: CGPoint, control: CGPoint)
```

```
    /// Adds a cubic Bézier curve to the
path, with the
    /// specified end point and control
points.
    ///
    /// This method constructs a curve
starting from the path's current
    /// point and ending at the specified
end point, with curvature
    /// defined by the two control
points. After this method appends
    /// that curve to the current path,
the end point of the curve
    /// becomes the path's current point.
    ///
    /// - Parameters:
    ///   the curve.
    ///   - control1: The first control
point of the curve, in user
    ///   space coordinates.
    ///   - control2: The second control
point of the curve, in user
    ///   space coordinates.
    ///
public mutating func addCurve(to end:
CGPoint, control1: CGPoint, control2:
CGPoint)
```

```
    /// Closes and completes the current
subpath.
    ///
    /// Appends a line from the current
point to the starting point of
```

```
    /// the current subpath and ends the
    subpath.
    /**
     /// After closing the subpath, your
     application can begin a new
     /// subpath without first calling
     `move(to:)`. In this case, a new
     /// subpath is implicitly created
     with a starting and current point
     /// equal to the previous subpath's
     starting point.
    /**
     public mutating func closeSubpath()

     /// Adds a rectangular subpath to the
     path.
    /**
     /// This is a convenience function
     that adds a rectangle to a path,
     /// starting by moving to the bottom-
     left corner and then adding
     /// lines counter-clockwise to create
     a rectangle, closing the
     /// subpath.
    /**
     /// - Parameters:
     ///   - rect: A rectangle, specified
     in user space coordinates.
     ///   - transform: An affine
     transform to apply to the rectangle
     ///   before adding to the path.
     Defaults to the identity
     ///   transform if not specified.
```

```
///  
public mutating func addRect(_ rect:  
CGRect, transform: CGAffineTransform  
= .identity)  
  
    /// Adds a rounded rectangle to the  
path.  
    ///  
    /// This is a convenience function  
that adds a rounded rectangle to  
    /// a path, starting by moving to the  
center of the right edge and  
    /// then adding lines and curves  
counter-clockwise to create a  
    /// rounded rectangle, closing the  
subpath.  
    ///  
    /// - Parameters:  
    ///   - rect: A rectangle, specified  
in user space coordinates.  
    ///   - cornerSize: The size of the  
corners, specified in user space  
    ///   coordinates.  
    ///   - style: The corner style.  
Defaults to the `continuous` style  
    /// if not specified.  
    /// - transform: An affine  
transform to apply to the rectangle  
    /// before adding to the path.  
Defaults to the identity  
    /// transform if not specified.  
    ///  
public mutating func
```

```
addRoundedRect(in rect: CGRect,  
cornerSize: CGSize, style:  
RoundedCornerStyle = .continuous,  
transform: CGAffineTransform = .identity)
```

```
    /// Adds a rounded rectangle with  
    uneven corners to the path.  
    ///  
    /// This is a convenience function  
    that adds a rounded rectangle to  
    /// a path, starting by moving to the  
    center of the right edge and  
    /// then adding lines and curves  
    counter-clockwise to create a  
    /// rounded rectangle, closing the  
    subpath.  
    ///  
    /// - Parameters:  
    ///   - rect: A rectangle, specified  
    in user space coordinates.  
    ///   - cornerRadii: The radius of  
    each corner of the rectangle,  
    ///     specified in user space  
    coordinates.  
    ///   - style: The corner style.  
    Defaults to the `continuous` style  
    ///     if not specified.  
    ///   - transform: An affine  
    transform to apply to the rectangle  
    ///     before adding to the path.  
    Defaults to the identity  
    ///     transform if not specified.  
    ///
```

```
    @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
    public mutating func  
addRoundedRect(in rect: CGRect,  
cornerRadii: RectangleCornerRadii, style:  
RoundedCornerStyle = .continuous,  
transform: CGAffineTransform = .identity)  
  
        /// Adds an ellipse that fits inside  
the specified rectangle to the  
        /// path.  
        ///  
        /// The ellipse is approximated by a  
sequence of Bézier  
        /// curves. Its center is the  
midpoint of the rectangle defined by  
        /// the `rect` parameter. If the  
rectangle is square, then the  
        /// ellipse is circular with a radius  
equal to one-half the width  
        /// (or height) of the rectangle. If  
the `rect` parameter specifies  
        /// a rectangular shape, then the  
major and minor axes of the  
        /// ellipse are defined by the width  
and height of the rectangle.  
        ///  
        /// The ellipse forms a complete  
subpath of the path—  
        /// that is, the ellipse drawing  
starts with a move-to operation  
        /// and ends with a close-subpath  
operation, with all moves
```

```
    /// oriented in the clockwise
    direction.
    /**
     * - Parameter:
     *   - rect: A rectangle that
     * defines the area for the ellipse to
     * fit in.
     * - transform: An affine
     * transform to apply to the ellipse
     * before adding to the path.
     * Defaults to the identity
     * transform if not specified.
    /**
     public mutating func addEllipse(in
rect: CGRect, transform:
CGAffineTransform = .identity)

    /// Adds a set of rectangular
    subpaths to the path.
    /**
     /// Calling this convenience method
is equivalent to repeatedly
     /// calling the
`addRect(_:transform:)` method for each
rectangle
     /// in the array.
    /**
     * - Parameter:
     *   - rects: An array of
rectangles, specified in user space
     * coordinates.
     * - transform: An affine
     * transform to apply to the ellipse
```

```
    /// before adding to the path.  
    Defaults to the identity  
    /// transform if not specified.  
    ///  
    public mutating func addRects(_  
rects: [CGRect], transform:  
CGAffineTransform = .identity)  
  
    /// Adds a sequence of connected  
straight-line segments to the path.  
    ///  
    /// Calling this convenience method  
is equivalent to applying the  
    /// transform to all points in the  
array, then calling the  
    /// `move(to:)` method with the first  
value in the `points` array,  
    /// then calling the `addLine(to:)`  
method for each subsequent  
    /// point until the array is  
exhausted. After calling this method,  
    /// the path's current point is the  
last point in the array.  
    ///  
    /// - Parameter:  
    /// - lines: An array of values  
that specify the start and end  
    /// points of the line segments  
to draw. Each point in the  
    /// array specifies a position in  
user space. The first point  
    /// in the array specifies the  
initial starting point.
```

```
    /// - transform: An affine
    transform to apply to the points
    /// before adding to the path.
    Defaults to the identity
    /// transform if not specified.
    ///
    public mutating func addLines(_
lines: [CGPoint])  
  
    /// Adds an arc of a circle to the
path, specified with a radius
    /// and a difference in angle.
    ///
    /// This method calculates starting
and ending points using the
    /// radius and angles you specify,
uses a sequence of cubic
    /// Bézier curves to approximate a
segment of a circle
    /// between those points, and then
appends those curves to the
    /// path.
    ///
    /// The `delta` parameter determines
both the length of the arc the
    /// direction in which the arc is
created; the actual direction of
    /// the final path is dependent on
the `transform` parameter and
    /// the current transform of a
context where the path is drawn.
    /// However, because SwiftUI by
default uses a vertically-flipped
```

```
    /// coordinate system (with the
    origin in the top-left of the
    /// view), specifying a clockwise arc
    results in a counterclockwise
    /// arc after the transformation is
    applied.

    ///
    /// If the path ends with an unclosed
    subpath, this method adds a
    /// line connecting the current point
    to the starting point of the
    /// arc. If there is no unclosed
    subpath, this method creates a new
    /// subpath whose starting point is
    the starting point of the arc.
    /// The ending point of the arc
    becomes the new current point of
    /// the path.

    ///
    /// - Parameters:
    ///   - center: The center of the
    arc, in user space coordinates.
    ///   - radius: The radius of the
    arc, in user space coordinates.
    ///   - startAngle: The angle to the
    starting point of the arc,
    ///     measured from the positive x-
    axis.
    ///   - delta: The difference between
    the starting angle and ending
    ///     angle of the arc. A positive
    value creates a counter-
    ///     clockwise arc (in user space
```

coordinates), and vice versa.

    /// - transform: An affine transform to apply to the arc before

    /// adding to the path. Defaults to the identity transform if

    /// not specified.

    ///

    public mutating func

addRelativeArc(center: CGPoint, radius: CGFloat, startAngle: Angle, delta: Angle, transform: CGAffineTransform = .identity)

    /// Adds an arc of a circle to the path, specified with a radius

    /// and angles.

    ///

    /// This method calculates starting and ending points using the

    /// radius and angles you specify, uses a sequence of cubic

    /// Bézier curves to approximate a segment of a circle

    /// between those points, and then appends those curves to the

    /// path.

    ///

    /// The `clockwise` parameter determines the direction in which the

    /// arc is created; the actual direction of the final path is

    /// dependent on the `transform` parameter and the current

    /// transform of a context where the

path is drawn. However,

```
    /// because SwiftUI by default uses a
    /// vertically-flipped coordinate
    /// system (with the origin in the
    top-left of the view),
    /// specifying a clockwise arc
    results in a counterclockwise arc
    /// after the transformation is
    applied.
```

///

```
    /// If the path ends with an unclosed
    subpath, this method adds a
    /// line connecting the current point
    to the starting point of the
    /// arc. If there is no unclosed
    subpath, this method creates a new
    /// subpath whose starting point is
    the starting point of the arc.
    /// The ending point of the arc
    becomes the new current point of
    /// the path.
```

///

```
    /// - Parameters:
    ///   - center: The center of the
    arc, in user space coordinates.
    ///   - radius: The radius of the
    arc, in user space coordinates.
    ///   - startAngle: The angle to the
    starting point of the arc,
    ///     measured from the positive x-
    axis.
    ///   - endAngle: The angle to the
    end point of the arc, measured
```

```
    ///      from the positive x-axis.  
    ///      - clockwise: true to make a  
clockwise arc; false to make a  
    ///      counterclockwise arc.  
    ///      - transform: An affine  
transform to apply to the arc before  
    ///      adding to the path. Defaults  
to the identity transform if  
    ///      not specified.  
    ///  
    public mutating func addArc(center:  
CGPoint, radius: CGFloat, startAngle:  
Angle, endAngle: Angle, clockwise: Bool,  
transform: CGAffineTransform = .identity)  
  
    /// Adds an arc of a circle to the  
path, specified with a radius  
    /// and two tangent lines.  
    ///  
    /// This method calculates two  
tangent lines—the first  
    /// from the current point to the  
tangent1End point, and the second  
    /// from the `tangent1End` point to  
the `tangent2End`  
    /// point—then calculates the start  
and end points for a  
    /// circular arc of the specified  
radius such that the arc is  
    /// tangent to both lines. Finally,  
this method approximates that  
    /// arc with a sequence of cubic  
Bézier curves and appends
```

```
    /// those curves to the path.  
    ///  
    /// If the starting point of the arc  
(that is, the point where a  
    /// circle of the specified radius  
must meet the first tangent line  
    /// in order to also be tangent to  
the second line) is not the  
    /// current point, this method  
appends a straight line segment from  
    /// the current point to the starting  
point of the arc.  
    ///  
    /// The ending point of the arc (that  
is, the point where a circle  
    /// of the specified radius must meet  
the second tangent line in  
    /// order to also be tangent to the  
first line) becomes the new  
    /// current point of the path.  
    ///  
    /// - Parameters:  
    ///   - tangent1End: The end point, in  
user space coordinates, for  
    ///     the first tangent line to be  
used in constructing the arc.  
    ///     (The start point for this  
tangent line is the path's  
    ///     current point.)  
    ///   - tangent2End: The end point,  
in user space coordinates, for  
    ///     the second tangent line to be  
used in constructing the arc.
```

```
    ///      (The start point for this
tangent line is the tangent1End
    ///      point.)
    ///      - radius: The radius of the
arc, in user space coordinates.
    ///      - transform: An affine
transform to apply to the arc before
    ///      adding to the path. Defaults
to the identity transform if
    ///      not specified.
    ///
public mutating func
addArc(tangent1End: CGPoint, tangent2End:
CGPoint, radius: CGFloat, transform:
CGAffineTransform = .identity)
```

```
    /// Appends another path value to
this path.
    ///
    /// If the `path` parameter is a non-
empty empty path, its elements
    /// are appended in order to this
path. Afterward, the start point
    /// and current point of this path
are those of the last subpath in
    /// the `path` parameter.
    ///
    /// - Parameters:
    ///     - path: The path to add.
    ///     - transform: An affine
transform to apply to the path
    ///     parameter before adding to
this path. Defaults to the
```

```
    /// identity transform if not
    specified.
    ///
    public mutating func addPath(_ path:
Path, transform: CGAffineTransform
= .identity)

    /// Returns the last point in the
path, or nil if the path contains
    /// no points.
    public var currentPoint: CGPoint? {
get }

    /// Returns a new weakly-simple copy
of this path.
    ///
    /// - Parameters:
    ///   - eoFill: Whether to use the
even-odd rule for determining
        /// which areas to treat as the
interior of the paths (if true),
        /// or the non-zero rule (if
false).
    /// - Returns: A new path.
    ///
    /// The returned path is a weakly-
simple path, has no
    /// self-intersections, and has a
normalized orientation. The
    /// result of filling this path using
either even-odd or non-zero
    /// fill rules is identical.
@available(iOS 17.0, macOS 14.0, tvOS
```

```
17.0, watchOS 10.0, *)
    public func normalized(eoFill: Bool = true) -> Path

        /// Returns a new path with filled
        regions common to both paths.
        ///
        /// - Parameters:
        ///   - other: The path to intersect.
        ///   - eoFill: Whether to use the
        even-odd rule for determining
            /// which areas to treat as the
            interior of the paths (if true),
            /// or the non-zero rule (if
            false).
        /// - Returns: A new path.
        ///
        /// The filled region of the
        resulting path is the overlapping area
        /// of the filled region of both
        paths. This can be used to clip
        /// the fill of a path to a mask.
        ///
        /// Any unclosed subpaths in either
        path are assumed to be closed.
        /// The result of filling this path
        using either even-odd or
        /// non-zero fill rules is identical.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public func intersection(_ other:
Path, eoFill: Bool = false) -> Path
```

```
    /// Returns a new path with filled
regions in either this path or
    /// the given path.
    ///
    /// - Parameters:
    ///   - other: The path to union.
    ///   - eoFill: Whether to use the
even-odd rule for determining
    ///           which areas to treat as the
interior of the paths (if true),
    ///           or the non-zero rule (if
false).
    /// - Returns: A new path.
    ///
    /// The filled region of resulting
path is the combination of the
    /// filled region of both paths added
together.
    ///
    /// Any unclosed subpaths in either
path are assumed to be closed.
    /// The result of filling this path
using either even-odd or
    /// non-zero fill rules is identical.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public func union(_ other: Path,
eoFill: Bool = false) -> Path

    /// Returns a new path with filled
regions from this path that are
    /// not in the given path.
    ///
```

```
    /// - Parameters:  
    ///   - other: The path to subtract.  
    ///   - eoFill: Whether to use the  
even-odd rule for determining  
    ///           which areas to treat as the  
interior of the paths (if true),  
    ///           or the non-zero rule (if  
false).  
    /// - Returns: A new path.  
    ///  
    /// The filled region of the  
resulting path is the filled region of  
    /// this path with the filled region  
'other' removed from it.  
    ///  
    /// Any unclosed subpaths in either  
path are assumed to be closed.  
    /// The result of filling this path  
using either even-odd or  
    /// non-zero fill rules is identical.  
    @available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
public func subtracting(_ other:  
Path, eoFill: Bool = false) -> Path  
  
    /// Returns a new path with filled  
regions either from this path or  
    /// the given path, but not in both.  
    ///  
    /// - Parameters:  
    ///   - other: The path to  
difference.  
    ///   - eoFill: Whether to use the
```

```
even-odd rule for determining
    ///      which areas to treat as the
interior of the paths (if true),
    ///      or the non-zero rule (if
false).
    /// - Returns: A new path.
    ///
    /// The filled region of the
resulting path is the filled region
    /// contained in either this path or
`other`, but not both.
    ///
    /// Any unclosed subpaths in either
path are assumed to be closed.
    /// The result of filling this path
using either even-odd or
    /// non-zero fill rules is identical.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public func symmetricDifference(_
other: Path, eoFill: Bool = false) ->
Path

    /// Returns a new path with a line
from this path that overlaps the
    /// filled regions of the given path.
    ///
    /// - Parameters:
    ///     - other: The path to intersect.
    ///     - eoFill: Whether to use the
even-odd rule for determining
    ///      which areas to treat as the
interior of the paths (if true),
```

```
    /// or the non-zero rule (if
false).
    /// - Returns: A new path.
    ///
    /// The line of the resulting path is
the line of this path that
    /// overlaps the filled region of
`other`.
    ///
    /// Intersected subpaths that are
clipped create open subpaths.
    /// Closed subpaths that do not
intersect `other` remain closed.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public func lineIntersection(_ other:
Path, eoFill: Bool = false) -> Path

    /// Returns a new path with a line
from this path that does not
    /// overlap the filled region of the
given path.
    ///
    /// - Parameters:
    ///   - other: The path to subtract.
    ///   - eoFill: Whether to use the
even-odd rule for determining
    ///           which areas to treat as the
interior of the paths (if true),
    ///           or the non-zero rule (if
false).
    /// - Returns: A new path.
    ///
```

```
    /// The line of the resulting path is
    /// the line of this path that
    /// does not overlap the filled
    /// region of `other`.
    ///
    /// Intersected subpaths that are
    /// clipped create open subpaths.
    /// Closed subpaths that do not
    /// intersect `other` remain closed.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public func lineSubtraction(_ other:
Path, eoFill: Bool = false) -> Path

    /// Returns a path constructed by
    /// applying the transform to all
    /// points of the path.
    ///
    /// - Parameter transform: An affine
    /// transform to apply to the path.
    ///
    /// - Returns: a new copy of the path
    /// with the transform applied to
    /// all points.
    ///
    public func applying(_ transform:
CGAffineTransform) -> Path

    /// Returns a path constructed by
    /// translating all its points.
    ///
    /// - Parameters:
    ///   - dx: The offset to apply in
```

the horizontal axis.

    /// - dy: The offset to apply in  
    the vertical axis.

    ///

    /// - Returns: a new copy of the path  
    with the offset applied to

    /// all points.

    ///

    public func offsetBy(dx: CGFloat, dy:  
    CGFloat) -> Path

}

@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, \*)

extension Path.Element : Sendable {

}

@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, \*)

extension Path.Element : BitwiseCopyable

{

}

    /// A schedule for updating a timeline  
    view at regular intervals.

    ///

    /// You can also use

    ``TimelineSchedule/periodic(from:by:)``

    to construct this

    /// schedule.

@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, \*)

public struct PeriodicTimelineSchedule :

```
TimelineSchedule, Sendable {

    /// The sequence of dates in periodic
schedule.
    ///
    /// The
``PeriodicTimelineSchedule/entries(from:m
ode:)`` method returns
    /// a value of this type, which is a
    ///
<doc://com.apple.documentation/documentat
ion/Swift/Sequence>
    /// of periodic dates in ascending
order. A ``TimelineView`` that you
    /// create updates its content at the
moments in time corresponding to the
    /// dates included in the sequence.
    public struct Entries : Sequence,
IteratorProtocol, Sendable {

        /// Advances to the next element
and returns it, or `nil` if no next
element
        /// exists.
        ///
        /// Repeatedly calling this
method returns, in order, all the
elements of the
        /// underlying sequence. As soon
as the sequence has run out of elements,
all
        /// subsequent calls return
`nil`.
```

```
    /**
     * You must not call this method
     * if any other copy of this iterator has
     * been
     *      /**
     * advanced with a call to its
     * `next()` method.
     */
     /**
     * The following example shows
     * how an iterator can be used explicitly to
     *      /**
     * emulate a `for`-`in` loop.
     * First, retrieve a sequence's iterator,
     * and
     *      /**
     * then call the iterator's
     * `next()` method until it returns `nil`.
     */
     /**
     *      let numbers = [2, 3, 5,
7]
     *      var numbersIterator =
numbers.makeIterator()
     /**
     *      while let num =
numbersIterator.next() {
     *          print(num)
     *      }
     *      // Prints "2"
     *      // Prints "3"
     *      // Prints "5"
     *      // Prints "7"
     *
     *      /**
     * - Returns: The next element
     * in the underlying sequence, if a next
     * element
     *      exists; otherwise, `nil`.
```

```
        public mutating func next() ->
Date? {
```

```
            /// A type representing the
sequence's elements.
            @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
            public typealias Element = Date
```

```
            /// A type that provides the
sequence's iteration interface and
            /// encapsulates its iteration
state.
            @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
            public typealias Iterator =
PeriodicTimelineSchedule.Entries
        }
```

```
        /// Creates a periodic update
schedule.
        ///
        /// Use the
``PeriodicTimelineSchedule/entries(from:m
ode:)`` method
        /// to get the sequence of dates.
        ///
        /// - Parameters:
        ///   - startDate: The date on which
to start the sequence.
        ///   - interval: The time interval
between successive sequence entries.
        public init(from startDate: Date, by
```

```
interval: TimeInterval)
```

```
    /// Provides a sequence of periodic  
    dates starting from around a given date.  
    ///  
    /// A ``TimelineView`` that you  
    create with a schedule calls this method  
    /// to ask the schedule when to  
    update its content. The method returns  
    /// a sequence of equally spaced  
    dates in increasing order that represent  
    /// points in time when the timeline  
    view should update.  
    ///  
    /// The schedule defines its  
    periodicity and phase alignment based on  
    the  
    /// parameters you pass to its  
    ``init(from:by:)`` initializer.  
    /// For example, for a `startDate`  
    and `interval` of `10:09:30` and  
    /// `60` seconds, the schedule  
    prepares to issue dates half past each  
    /// minute. The `startDate` that you  
    pass to the `entries(from:mode:)`  
    /// method then dictates the first  
    date of the sequence as the beginning of  
    /// the interval that the start date  
    overlaps. Continuing the example above,  
    /// a start date of `10:34:45` causes  
    the first sequence entry to be  
    /// `10:34:30`, because that's the  
    start of the interval in which the
```

```
    /// start date appears.
    public func entries(from startDate: Date, mode: TimelineScheduleMode) -> PeriodicTimelineSchedule.Entries
}

/// A container that animates its content by automatically cycling through
/// a collection of phases that you provide, each defining a discrete step
/// within an animation.
///
/// Use one of the phase animator view modifiers like
///
``View/phaseAnimator(_:content:animation:)`` to create a phased animation
/// in your app.
@available(iOS 17.0, macOS 14.0, tvOS 17.0, watchOS 10.0, *)
public struct PhaseAnimator<Phase, Content> : View where Phase : Equatable, Content : View {

    /// Cycles through a sequence of phases in response to changes in a
    /// specified value, animating updates to a view on each phase change.
    ///
    /// When the phase animator first appears, this initializer renders the
    /// `content` closure using the first phase as input to the closure.
```

```
    /// When the value of the `trigger`  
input changes, the animator  
    /// reevaluates the `content` closure  
using the value from the second phase  
    /// and animates the change. This  
procedure repeats with each  
    /// successive phase until reaching  
the last phase, at which point  
    /// the animator loops back to the  
first phase.  
    ////  
    /// - Parameters:  
    /// - phases: The sequence of  
phases to cycle through. Ensure that the  
    /// sequence isn't empty. If it  
is, SwiftUI logs a runtime warning and  
    /// also returns a visual warning  
as the output view.  
    /// - trigger: A value whose  
changes cause the animator to use the  
    /// next phase.  
    /// - content: A view builder  
closure that takes the current phase as  
an  
    /// input. Return a view that's  
based on the phase input.  
    /// - animation: A closure that  
takes the current phase as input. Return  
    /// the animation to use when  
transitioning to the next phase. If you  
    /// return `nil`, the transition  
doesn't animate. If you don't set this  
    /// parameter, SwiftUI uses a
```

default animation.

```
    public init(_ phases: some Sequence<Phase>, trigger: some Equatable, @ViewBuilder content: @escaping (Phase) -> Content, animation: @escaping (Phase) -> Animation? = { _ in .default })
```

/// Cycles through a sequence of phases continuously, animating updates to  
 /// a view on each phase change.

///

/// When the phase animator first appears, this initializer renders the  
 /// `content` closure using the first phase as input to the closure.

/// The animator then begins immediately animating to the view produced by

/// sending the second phase to the `content` closure using

/// the animation returned from the `animation` closure. This procedure

/// repeats for successive phases until reaching the last phase, after which

/// the animator loops back to the first phase again.

///

/// - Parameters:

/// - phases: The sequence of phases to cycle through. Ensure that the

/// sequence isn't empty. If it is, SwiftUI logs a runtime warning and

```
    ///      also returns a visual warning  
    as the output view.
```

```
    /// - content: A view builder  
closure that takes the current phase as  
an
```

```
    ///      input. Return a view that's  
based on the current phase.
```

```
    /// - animation: A closure that  
takes the current phase as input. Return
```

```
    ///      the animation to use when  
transitioning to the next phase. If you
```

```
    ///      return `nil`, the transition  
doesn't animate. If you don't set this
```

```
    ///      parameter, SwiftUI uses a  
default animation.
```

```
public init(_ phases: some  
Sequence<Phase>, @ViewBuilder content:  
@escaping (Phase) -> Content, animation:  
@escaping (Phase) -> Animation? = { _  
in .default })
```

```
    /// The content and behavior of the  
view.
```

```
    ///
```

```
    /// When you implement a custom view,  
you must implement a computed
```

```
    /// `body` property to provide the  
content for your view. Return a view
```

```
    /// that's composed of built-in views  
that SwiftUI provides, plus other
```

```
    /// composite views that you've  
already defined:
```

```
    ///
```

```
///     struct MyView: View {
///         var body: some View {
///             Text("Hello, World!")
///         }
///     }
///
/// For more information about
composing views and a view hierarchy,
/// see <doc:Declaring-a-Custom-
View>.
@MainActor @preconcurrency public var
body: some View { get }

/// The type of view representing the
body of this view.
///
/// When you create a custom view,
Swift infers this type from your
/// implementation of the required
``View/body-swift.property`` property.
@available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
public typealias Body = some View
}

/// A set of view types that may be
pinned to the bounds of a scroll view.
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
public struct PinnedScrollableViews :
OptionSet, Sendable {

    /// The corresponding value of the
```

raw type.

```
///  
/// A new instance initialized with  
`rawValue` will be equivalent to this  
/// instance. For example:  
///  
///     enum PaperSize: String {  
///         case A4, A5, Letter,  
Legal  
///     }  
///  
///     let selectedSize =  
PaperSize.Letter  
///     print(selectedSize.rawValue)  
///     // Prints "Letter"  
///  
///     print(selectedSize ==  
PaperSize(rawValue:  
selectedSize.rawValue)!)  
///     // Prints "true"  
public let rawValue: UInt32  
  
    /// Creates a new option set from the  
given raw value.  
    ///  
    /// This initializer always succeeds,  
even if the value passed as `rawValue`  
    /// exceeds the static properties  
declared as part of the option set. This  
    /// example creates an instance of  
`ShippingOptions` with a raw value beyond  
    /// the highest element, with a bit  
mask that effectively contains all the
```

```
    /// declared static members.  
    ///  
    ///     let extraOptions =  
ShippingOptions(rawValue: 255)  
    ///  
print(extraOptions.isStrictSuperset(of: .  
all))  
    ///     // Prints "true"  
    ///  
    /// - Parameter rawValue: The raw  
value of the option set to create. Each  
bit  
    ///     of `rawValue` potentially  
represents an element of the option set,  
    ///     though raw values may include  
bits that are not defined as distinct  
    ///     values of the `OptionSet` type.  
public init(rawValue: UInt32)  
  
    /// The header view of each `Section`  
will be pinned.  
public static let sectionHeaders:  
PinnedScrollableViews  
  
    /// The footer view of each `Section`  
will be pinned.  
public static let sectionFooters:  
PinnedScrollableViews  
  
    /// The type of the elements of an  
array literal.  
@available(iOS 14.0, tvOS 14.0,  
watchOS 7.0, macOS 11.0, *)
```

```
    public typealias ArrayLiteralElement
= PinnedScrollableViews

        /// The element type of the option
set.

        ///
        /// To inherit all the default
implementations from the `OptionSet`
protocol,
        /// the `Element` type must be
`Self`, the default.
        @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
        public typealias Element =
PinnedScrollableViews

        /// The raw type that can be used to
represent all values of the conforming
        /// type.
        ///
        /// Every distinct value of the
conforming type has a corresponding
unique
        /// value of the `RawValue` type, but
there may be values of the `RawValue`'
        /// type that don't have a
corresponding value of the conforming
type.
        @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
        public typealias RawValue = UInt32
}
```

```
/// A placeholder used to construct an
/// inline modifier, transition, or other
/// helper type.
///
/// You don't use this type directly.
/// Instead SwiftUI creates this type on
/// your behalf.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct
PlaceholderContentView<Value> : View {

    /// The type of view representing the
    body of this view.
    ///
    /// When you create a custom view,
    Swift infers this type from your
    /// implementation of the required
    ``View/body-swift.property`` property.
    @available(iOS 17.0, tvOS 17.0,
    watchOS 10.0, macOS 14.0, *)
    public typealias Body = Never
}

/// A named value produced by a view.
///
/// A view with multiple children
/// automatically combines its values for a
/// given
/// preference into a single value
/// visible to its ancestors.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
```

```
public protocol PreferenceKey {  
  
    /// The type of value produced by  
    this preference.  
    associatedtype Value  
  
    /// The default value of the  
    preference.  
    ///  
    /// Views that have no explicit value  
    for the key produce this default  
    /// value. Combining child views may  
    remove an implicit value produced by  
    /// using the default. This means  
    that `reduce(value: &x, nextValue:  
    /// {defaultValue})` shouldn't change  
    the meaning of `x`.  
    static var defaultValue: Self.Value {  
        get }  
  
    /// Combines a sequence of values by  
    modifying the previously-accumulated  
    /// value with the result of a  
    closure that provides the next value.  
    ///  
    /// This method receives its values  
    in view-tree order. Conceptually, this  
    /// combines the preference value  
    from one tree with that of its next  
    /// sibling.  
    ///  
    /// - Parameters:  
    ///     - value: The value accumulated
```

```
through previous calls to this method.  
    /// The implementation should  
    modify this value.  
    /// - nextValue: A closure that  
    returns the next value in the sequence.  
    static func reduce(value: inout  
Self.Value, nextValue: () -> Self.Value)  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension PreferenceKey where  
Self.Value : ExpressibleByNilLiteral {  
  
    /// Let nil-expressible values  
    default-initialize to nil.  
    public static var defaultValue:  
Self.Value { get }  
}  
  
/// A key for specifying the preferred  
color scheme.  
///  
/// Don't use this key directly. Instead,  
set a preferred color scheme for a  
/// view using the  
``View/preferredColorScheme(_:)`` view  
modifier. Get the  
/// current color scheme for a view by  
accessing the  
/// ``EnvironmentValues/colorScheme``  
value.  
@available(iOS 13.0, macOS 11.0, tvOS
```

```
13.0, watchOS 6.0, *)
public struct PreferredColorSchemeKey : PreferenceKey {

    /// The type of value produced by this preference.
    public typealias Value = ColorScheme?

    /// Combines a sequence of values by modifying the previously-accumulated
    /// value with the result of a closure that provides the next value.
    ///
    /// This method receives its values in view-tree order. Conceptually, this
    /// combines the preference value from one tree with that of its next
    /// sibling.
    ///
    /// - Parameters:
    ///   - value: The value accumulated through previous calls to this method.
    ///   - The implementation should modify this value.
    ///   - nextValue: A closure that returns the next value in the sequence.
    public static func reduce(value: inout PreferredColorSchemeKey.Value,
                             nextValue: () -> PreferredColorSchemeKey.Value)
}

@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
@frozen public struct ProjectionTransform
{

    public var m11: CGFloat
    public var m12: CGFloat
    public var m13: CGFloat
    public var m21: CGFloat
    public var m22: CGFloat
    public var m23: CGFloat
    public var m31: CGFloat
    public var m32: CGFloat
    public var m33: CGFloat

    @inlinable public init()
    @inlinable public init(_ m: CGAffineTransform)
    @inlinable public init(_ m: CATransform3D)

    @inlinable public var isIdentity: Bool { get }
```

```
    @inlinable public var isAffine: Bool
{ get }

    public mutating func invert() -> Bool

    public func inverted() ->
ProjectionTransform
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ProjectionTransform : Equatable
{

    /// 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:
ProjectionTransform, b:
ProjectionTransform) -> Bool
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
```

```
extension ProjectionTransform {  
  
    @inlinable public func  
concatenating(_ rhs: ProjectionTransform)  
-> ProjectionTransform  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ProjectionTransform : Sendable  
{  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ProjectionTransform :  
BitwiseCopyable {  
}  
  
/// A type indicating the prominence of a  
view hierarchy.  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
public enum Prominence : Sendable {  
  
    /// The standard prominence.  
    case standard  
  
    /// An increased prominence.  
    ///  
    /// - Note: Not all views will react  
to increased prominence.  
    case increased
```

```
    /// 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: Prominence,  
b: Prominence) -> Bool
```

```
    /// 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)
```

```
    /// 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 }
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Prominence : Equatable {  
}
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Prominence : Hashable {  
}  
  
/// A proposal for the size of a view.  
///  
/// During layout in SwiftUI, views  
choose their own size, but they do that  
/// in response to a size proposal from  
their parent view. When you create  
/// a custom layout using the ``Layout``  
protocol, your layout container  
/// participates in this process using  
`ProposedViewSize` instances.  
/// The layout protocol's methods take a  
proposed size input that you  
/// can take into account when arranging  
views and calculating the size of  
/// the composite container. Similarly,  
your layout proposes a size to each  
/// of its own subviews when it measures  
and places them.  
///  
/// Layout containers typically measure  
their subviews by proposing several  
/// sizes and looking at the responses.  
The container can use this information  
/// to decide how to allocate space among  
its subviews. A  
/// layout might try the following  
special proposals:  
///
```

```
/// * The ``zero`` proposal; the view  
responds with its minimum size.  
/// * The ``infinity`` proposal; the view  
responds with its maximum size.  
/// * The ``unspecified`` proposal; the  
view responds with its ideal size.  
///  
/// A layout might also try special cases  
for one dimension at a time. For  
/// example, an ``HStack`` might measure  
the flexibility of its subviews'  
/// widths, while using a fixed value for  
the height.  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
@frozen public struct ProposedViewSize :  
Equatable {  
  
    /// The proposed horizontal size  
measured in points.  
    ///  
    /// A value of `nil` represents an  
unspecified width proposal, which a view  
    /// interprets to mean that it should  
use its ideal width.  
    public var width: CGFloat?  
  
    /// The proposed vertical size  
measured in points.  
    ///  
    /// A value of `nil` represents an  
unspecified height proposal, which a view  
    /// interprets to mean that it should
```

use its ideal height.

```
public var height: CGFloat?
```

```
    /// A size proposal that contains  
zero in both dimensions.
```

```
    ///
```

```
    /// Subviews of a custom layout  
return their minimum size when you  
propose
```

```
    /// this value using the  
``LayoutSubview/dimensions(in:)`` method.
```

```
    /// A custom layout should also  
return its minimum size from the
```

```
    ///
```

```
``Layout/sizeThatFits(proposal:subviews:c  
ache:)`` method for this
```

```
    /// value.
```

```
public static let zero:  
ProposedViewSize
```

```
    /// The proposed size with both  
dimensions left unspecified.
```

```
    ///
```

```
    /// Both dimensions contain `nil` in  
this size proposal.
```

```
    /// Subviews of a custom layout  
return their ideal size when you propose
```

```
    /// this value using the
```

```
``LayoutSubview/dimensions(in:)`` method.
```

```
    /// A custom layout should also  
return its ideal size from the
```

```
    ///
```

```
``Layout/sizeThatFits(proposal:subviews:c
```

```
ache:``` method for this
    /// value.
    public static let unspecified:
ProposedViewSize

    /// A size proposal that contains
infinity in both dimensions.
    ///
    /// Both dimensions contain
    ///
<doc://com.apple.documentation/documentation/CoreFoundation/CGFloat/1454161-
infinity>
    /// in this size proposal.
    /// Subviews of a custom layout
return their maximum size when you
propose
    /// this value using the
``LayoutSubview/dimensions(in:)`` method.
    /// A custom layout should also
return its maximum size from the
    ///
``Layout/sizeThatFits(proposal:subviews:c
ache:)``` method for this
    /// value.
    public static let infinity:
ProposedViewSize

    /// Creates a new proposed size using
the specified width and height.
    ///
    /// - Parameters:
    ///   - width: A proposed width in
```

```
points. Use a value of `nil` to indicate
    ///      that the width is unspecified
for this proposal.
    ///      - height: A proposed height in
points. Use a value of `nil` to
    ///      indicate that the height is
unspecified for this proposal.
@inlinable public init(width:
CGFloat?, height: CGFloat?)

    /// Creates a new proposed size from
a specified size.
    ///
    /// - Parameter size: A proposed size
with dimensions measured in points.
@inlinable public init(_ size:
CGSize)

    /// Creates a new proposal that
replaces unspecified dimensions in this
    /// proposal with the corresponding
dimension of the specified size.
    ///
    /// Use the default value to prevent
a flexible view from disappearing
    /// into a zero-sized frame, and
ensure the unspecified value remains
    /// visible during debugging.
    ///
    /// - Parameter size: A set of
concrete values to use for the size
proposal
    ///      in place of any unspecified
```

```
dimensions. The default value is `10`  
    /// for both dimensions.  
    ///  
    /// - Returns: A new, fully specified  
size proposal.  
    @inlinable public func  
replacingUnspecifiedDimensions(by size:  
CGSize = CGSize(width: 10, height: 10))  
-> CGSize  
  
    /// 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:  
ProposedViewSize, b: ProposedViewSize) ->  
Bool  
}  
  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension ProposedViewSize : Sendable {  
}  
  
@available(iOS 16.0, macOS 13.0, tvOS
```

```
16.0, watchOS 9.0, *)
extension ProposedViewSize :  
BitwiseCopyable {  
}  
  
/// A transition that when added to a  
view will animate the view's insertion by  
/// moving it in from the specified edge  
while fading it in, and animate its  
/// removal by moving it out towards the  
opposite edge and fading it out.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
@MainActor @preconcurrency public struct  
PushTransition : Transition {  
  
    /// The edge from which the view will  
be animated in.  
    @MainActor @preconcurrency public var  
edge: Edge  
  
    /// Creates a transition that  
animates a view by moving and fading it.  
    @MainActor @preconcurrency public  
init(edge: Edge)  
  
    /// Gets the current body of the  
caller.  
    ///  
    /// `content` is a proxy for the view  
that will have the modifier  
    /// represented by `Self` applied to  
it.
```

```
    @MainActor @preconcurrency public
func body(content:
PushTransition.Content, phase:
TransitionPhase) -> some View

        /// The type of view representing the
body.
        @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
        public typealias Body = some View
}

/// A radial gradient.
///
/// The gradient applies the color
function as the distance from a center
/// point, scaled to fit within the
defined start and end radii. The
/// gradient maps the unit space center
point into the bounding rectangle of
/// each shape filled with the gradient.
///
/// When using a radial gradient as a
shape style, you can also use
```
``ShapeStyle/radialGradient(_:center:startRadius:endRadius:)``.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct RadialGradient : ShapeStyle, View, Sendable {
```

```
    /// Creates a radial gradient from a
    base gradient.
    public init(gradient: Gradient,
center: UnitPoint, startRadius: CGFloat,
endRadius: CGFloat)

    /// Creates a radial gradient from a
collection of colors.
    public init(colors: [Color], center:
UnitPoint, startRadius: CGFloat,
endRadius: CGFloat)

    /// Creates a radial gradient from a
collection of color stops.
    public init(stops: [Gradient.Stop],
center: UnitPoint, startRadius: CGFloat,
endRadius: CGFloat)

    /// The type of view representing the
body of this view.
    /**
     /// When you create a custom view,
Swift infers this type from your
     /// implementation of the required
``View/body-swift.property`` property.
     @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Body

    /// The type of shape style this will
resolve to.
    /**
     /// When you create a custom shape
```

```
style, Swift infers this type
    /// from your implementation of the
    required `resolve` function.
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Resolved = Never
}

/// A rectangular shape aligned inside
the frame of the view containing it.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct Rectangle : Shape {

    /// Describes this shape as a path
    within a rectangular frame of reference.
    ///
    /// - Parameter rect: The frame of
    reference for describing this shape.
    ///
    /// - Returns: A path that describes
    this shape.
    nonisolated public func path(in rect:
CGRect) -> Path

    /// Returns the behavior this shape
    should use for different layout
    /// directions.
    ///
    /// If the layoutDirectionBehavior
    for a Shape is one that mirrors, the
        /// shape's path will be mirrored
    horizontally when in the specified layout
```

```
    /// direction. When mirrored, the
    individual points of the path will be
    /// transformed.
    ///
    /// Defaults to `mirrors` when
    deploying on iOS 17.0, macOS 14.0,
    /// tvOS 17.0, watchOS 10.0 and
    later, and to `fixed` if not.
    /// To mirror a path when deploying
    to earlier releases, either use
    ///
`View.flipsForRightToLeftLayoutDirection`
for a filled or stroked
    /// shape or conditionally mirror the
    points in the path of the shape.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    nonisolated public var
layoutDirectionBehavior:
LayoutDirectionBehavior { get }

    /// Creates a new rectangle shape.
    @inlinable public init()

    /// The type defining the data to
    animate.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias AnimatableData =
EmptyAnimatableData

    /// The type of view representing the
    body of this view.
```

```
///  
/// When you create a custom view,  
Swift infers this type from your  
/// implementation of the required  
``View/body-swift.property`` property.  
@available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
public typealias Body  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Rectangle : InsettableShape {  
  
    /// Returns `self` inset by `amount`.  
    @inlinable nonisolated public func  
    inset(by amount: CGFloat) -> some  
    InsettableShape  
  
    /// The type of the inset shape.  
    @available(iOS 13.0, tvOS 13.0,  
    watchOS 6.0, macOS 10.15, *)  
    public typealias InsetShape = some  
    InsettableShape  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Rectangle : BitwiseCopyable {  
}  
  
/// Describes the corner radius values of
```

```
a rounded rectangle with
/// uneven corners.
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
@frozen public struct
RectangleCornerRadii : Equatable,
Animatable {

    /// The radius of the top-leading
    corner.
    public var topLeading: CGFloat

    /// The radius of the bottom-leading
    corner.
    public var bottomLeading: CGFloat

    /// The radius of the bottom-trailing
    corner.
    public var bottomTrailing: CGFloat

    /// The radius of the top-trailing
    corner.
    public var topTrailing: CGFloat

    /// Creates a new set of corner radii
    for a rounded rectangle with
    /// uneven corners.
    ///
    /// - Parameters:
    ///   - topLeading: the radius of the
    top-leading corner.
    ///   - bottomLeading: the radius of
    the bottom-leading corner.
```

```
    /// - bottomTrailing: the radius of
the bottom-trailing corner.
    /// - topTrailing: the radius of
the top-trailing corner.
    public init(topLeading: CGFloat = 0,
bottomLeading: CGFloat = 0,
bottomTrailing: CGFloat = 0, topTrailing:
CGFloat = 0)

    /// The type defining the data to
animate.
    public typealias AnimatableData =
AnimatablePair<AnimatablePair<CGFloat,
CGFloat>, AnimatablePair<CGFloat,
CGFloat>>

    /// The data to animate.
    public var animatableData:
RectangleCornerRadii.AnimatableData

    /// 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:
```

```
RectangleCornerRadii, b:  
RectangleCornerRadii) -> Bool  
}  
  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension RectangleCornerRadii :  
BitwiseCopyable {  
}  
  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension RectangleCornerRadii : Sendable  
{  
}  
  
/// The reasons to apply a redaction to  
data displayed on screen.  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
public struct RedactionReasons :  
OptionSet, Sendable {  
  
    /// The raw value.  
    public let rawValue: Int  
  
    /// Creates a new set from a raw  
value.  
    ///  
    /// - Parameter rawValue: The raw  
value with which to create the  
    ///   reasons for redaction.  
    public init(rawValue: Int)
```

```
    /// Displayed data should appear as
    generic placeholders.
    ///
    /// Text and images will be
    automatically masked to appear as
    /// generic placeholders, though
    maintaining their original size and
    shape.
    /// Use this to create a placeholder
    UI without directly exposing
    /// placeholder data to users.
    public static let placeholder:
RedactionReasons
```

```
    /// Displayed data should be obscured
    to protect private information.
    ///
    /// Views marked with
    `privacySensitive` will be automatically
    redacted
    /// using a standard styling. To
    apply a custom treatment the redaction
    /// reason can be read out of the
    environment.
    ///
    ///     struct BankingContentView:
View {
    ///
    @Environment(\.redactionReasons) var
    redactionReasons
    ///
    ///         var body: some View {
```

```
    /**
     * If redactionReasons contains .privacy, return FullAppCover()
     * otherwise, return AppContent()
     */
    @available(iOS 15.0, macOS 12.0, tvOS 15.0, watchOS 8.0, *)
    public static let privacy: RedactionReasons
```

```
    /// Displayed data should appear as
    invalidated and pending a new update.
    ///
    /// Views marked with
    `invalidatableContent` will be
    automatically
    /// redacted with a standard styling
    indicating the content is invalidated
    /// and new content will be available
    soon.
    @available(iOS 17.0, macOS 14.0, tvOS
    17.0, watchOS 10.0, *)
    public static let invalidated:
    RedactionReasons
```

```
    /// The type of the elements of an
    array literal.
    @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
public typealias ArrayLiteralElement
```

```
= RedactionReasons
```

```
    /// The element type of the option
    set.
    ///
    /// To inherit all the default
    implementations from the `OptionSet`
    protocol,
    /// the `Element` type must be
    `Self`, the default.
    @available(iOS 14.0, tvOS 14.0,
    watchOS 7.0, macOS 11.0, *)
    public typealias Element =
RedactionReasons
```

```
    /// The raw type that can be used to
    represent all values of the conforming
    /// type.
    ///
    /// Every distinct value of the
    conforming type has a corresponding
    unique
    /// value of the `RawValue` type, but
    there may be values of the `RawValue`-
    /// type that don't have a
    corresponding value of the conforming
    type.
    @available(iOS 14.0, tvOS 14.0,
    watchOS 7.0, macOS 11.0, *)
    public typealias RawValue = Int
}
```

```
/// A shape with a rotation transform
```

```
        applied to it.  
    @available(iOS 13.0, macOS 10.15, tvOS  
    13.0, watchOS 6.0, *)  
    @frozen public struct  
    RotatedShape<Content> : Shape where  
    Content : Shape {  
  
        public var shape: Content  
  
        public var angle: Angle  
  
        public var anchor: UnitPoint  
  
        @inlinable public init(shape:  
        Content, angle: Angle, anchor: UnitPoint  
        = .center)  
  
            /// Describes this shape as a path  
            within a rectangular frame of reference.  
            ///  
            /// - Parameter rect: The frame of  
            /// reference for describing this shape.  
            ///  
            /// - Returns: A path that describes  
            /// this shape.  
        nonisolated public func path(in rect:  
        CGRect) -> Path  
  
            /// An indication of how to style a  
            /// shape.  
            ///  
            /// SwiftUI looks at a shape's role  
            /// when deciding how to apply a
```

```
    /// ``ShapeStyle`` at render time.  
The ``Shape`` protocol provides a  
    /// default implementation with a  
value of ``ShapeRole/fill``. If you  
    /// create a composite shape, you can  
provide an override of this property  
    /// to return another value, if  
appropriate.  
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    nonisolated public static var role:  
ShapeRole { get }  
  
    /// Returns the behavior this shape  
should use for different layout  
    /// directions.  
    ///  
    /// If the layoutDirectionBehavior  
for a Shape is one that mirrors, the  
    /// shape's path will be mirrored  
horizontally when in the specified layout  
    /// direction. When mirrored, the  
individual points of the path will be  
    /// transformed.  
    ///  
    /// Defaults to `mirrors` when  
deploying on iOS 17.0, macOS 14.0,  
    /// tvOS 17.0, watchOS 10.0 and  
later, and to `fixed` if not.  
    /// To mirror a path when deploying  
to earlier releases, either use  
    ///  
`View.flipsForRightToLeftLayoutDirection`
```

```
for a filled or stroked
    /// shape or conditionally mirror the
    points in the path of the shape.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    nonisolated public var
layoutDirectionBehavior:
LayoutDirectionBehavior { get }

    /// The type defining the data to
animate.
    public typealias AnimatableData =
AnimatablePair<Content.AnimatableData,
AnimatablePair<Angle.AnimatableData,
UnitPoint.AnimatableData>>

    /// The data to animate.
    public var animatableData:
RotatedShape<Content>.AnimatableData

    /// The type of view representing the
body of this view.
    /**
     /// When you create a custom view,
Swift infers this type from your
     /// implementation of the required
``View/body-swift.property`` property.
     @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
     public typealias Body
}

@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
extension RotatedShape : InsettableShape
where Content : InsettableShape {

    /// Returns `self` inset by `amount`.
    @inlinable nonisolated public func
    inset(by amount: CGFloat) ->
    RotatedShape<Content.InsetShape>

    /// The type of the inset shape.
    @available(iOS 13.0, tvOS 13.0,
    watchOS 6.0, macOS 10.15, *)
    public typealias InsetShape =
    RotatedShape<Content.InsetShape>
}

/// Defines the shape of a rounded
rectangle's corners.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public enum RoundedCornerStyle : Sendable
{

    /// Quarter-circle rounded rect
    corners.
    case circular

    /// Continuous curvature rounded rect
    corners.
    case continuous

    /// 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:  
RoundedCornerStyle, b:  
RoundedCornerStyle) -> Bool  
  
    /// 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)  
  
    /// 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 }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension RoundedCornerStyle : Equatable  
{  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
extension RoundedCornerStyle : Hashable {
}

/// A rectangular shape with rounded
corners, aligned inside the frame of the
/// view containing it.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct RoundedRectangle : Shape {

    /// The width and height of the
    rounded rectangle's corners.
    public var cornerSize: CGSize

    /// The style of corners drawn by the
    rounded rectangle.
    public var style: RoundedCornerStyle

    /// Creates a new rounded rectangle
    shape.
    ///
    /// - Parameters:
    ///   - cornerSize: the width and
    height of the rounded corners.
    ///   - style: the style of corners
    drawn by the shape.
    @inlinable public init(cornerSize:
    CGSize, style: RoundedCornerStyle
    = .continuous)

    /// Creates a new rounded rectangle
```

shape.

///

/// – Parameters:

/// – cornerRadius: the radius of  
the rounded corners.

/// – style: the style of corners  
drawn by the shape.

**@inlinable public init**(cornerRadius:  
**CGFloat**, style: **RoundedCornerStyle**  
= .continuous)

/// Describes this shape as a path  
within a rectangular frame of reference.

///

/// – Parameter rect: The frame of  
reference for describing this shape.

///

/// – Returns: A path that describes  
this shape.

**nonisolated public func** path(in rect:  
**CGRect**) -> **Path**

/// Returns the behavior this shape  
should use for different layout

/// directions.

///

/// If the layoutDirectionBehavior  
for a Shape is one that mirrors, the

/// shape's path will be mirrored  
horizontally when in the specified layout

/// direction. When mirrored, the  
individual points of the path will be

/// transformed.

```
///  
/// Defaults to `mirrors` when  
deploying on iOS 17.0, macOS 14.0,  
/// tvOS 17.0, watchOS 10.0 and  
later, and to `fixed` if not.  
/// To mirror a path when deploying  
to earlier releases, either use  
///  
`View.flipsForRightToLeftLayoutDirection`  
for a filled or stroked  
/// shape or conditionally mirror the  
points in the path of the shape.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
nonisolated public var  
layoutDirectionBehavior:  
LayoutDirectionBehavior { get }  
  
/// The data to animate.  
public var animatableData:  
CGSize.AnimatableData  
  
/// The type defining the data to  
animate.  
@available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
public typealias AnimatableData =  
CGSize.AnimatableData  
  
/// The type of view representing the  
body of this view.  
///  
/// When you create a custom view,
```

```
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Body
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension RoundedRectangle :
InsettableShape {

    /// Returns `self` inset by `amount`.
    @inlinable nonisolated public func
inset(by amount: CGFloat) -> some
InsettableShape

    /// The type of the inset shape.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias InsetShape = some
InsettableShape
}

/// A set of symbolic safe area regions.
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
@frozen public struct SafeAreaRegions : OptionSet {

    /// The corresponding value of the
```

raw type.

```
///  
/// A new instance initialized with  
`rawValue` will be equivalent to this  
/// instance. For example:  
///  
///     enum PaperSize: String {  
///         case A4, A5, Letter,  
Legal  
///     }  
///  
///     let selectedSize =  
PaperSize.Letter  
///     print(selectedSize.rawValue)  
///     // Prints "Letter"  
///  
///     print(selectedSize ==  
PaperSize(rawValue:  
selectedSize.rawValue)!)  
///     // Prints "true"  
public let rawValue: UInt  
  
/// Creates a new option set from the  
given raw value.  
///  
/// This initializer always succeeds,  
even if the value passed as `rawValue`  
/// exceeds the static properties  
declared as part of the option set. This  
/// example creates an instance of  
`ShippingOptions` with a raw value beyond  
/// the highest element, with a bit  
mask that effectively contains all the
```

```
    /// declared static members.  
    ///  
    ///     let extraOptions =  
ShippingOptions(rawValue: 255)  
    ///  
print(extraOptions.isStrictSuperset(of: .  
all))  
    ///     // Prints "true"  
    ///  
    /// - Parameter rawValue: The raw  
value of the option set to create. Each  
bit  
    ///     of `rawValue` potentially  
represents an element of the option set,  
    ///     though raw values may include  
bits that are not defined as distinct  
    ///     values of the `OptionSet` type.  
    @inlinable public init(rawValue:  
UInt)
```

```
    /// The safe area defined by the  
device and containers within the  
    /// user interface, including  
elements such as top and bottom bars.  
    public static let container:  
SafeAreaRegions
```

```
    /// The safe area matching the  
current extent of any software  
    /// keyboard displayed over the view  
content.  
    public static let keyboard:  
SafeAreaRegions
```

```
    /// All safe area regions.
    public static let all:
SafeAreaRegions

    /// The type of the elements of an
array literal.
    @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
    public typealias ArrayLiteralElement
= SafeAreaRegions

    /// The element type of the option
set.
    /**
     /// To inherit all the default
implementations from the `OptionSet`
protocol,
     /// the `Element` type must be
`Self`, the default.
     @available(iOS 14.0, tvOS 14.0,
watchOS 7.0, macOS 11.0, *)
     public typealias Element =
SafeAreaRegions

    /// The raw type that can be used to
represent all values of the conforming
    /// type.
    /**
     /// Every distinct value of the
conforming type has a corresponding
unique
     /// value of the `RawValue` type, but
```

```
there may be values of the `RawValue`  
    /// type that don't have a  
corresponding value of the conforming  
type.  
    @available(iOS 14.0, tvOS 14.0,  
watchOS 7.0, macOS 11.0, *)  
    public typealias RawValue = UInt  
}  
  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
extension SafeAreaRegions : Sendable {  
}  
  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
extension SafeAreaRegions :  
BitwiseCopyable {  
}  
  
/// Returns a transition that scales the  
view.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
@MainActor @preconcurrency public struct  
ScaleTransition : Transition {  
  
    /// The amount to scale the view by.  
    @MainActor @preconcurrency public var  
scale: Double  
  
    /// The anchor point to scale the  
view around.
```

```
    @MainActor @preconcurrency public var
anchor: UnitPoint

    /// Creates a transition that scales
the view by the specified amount.
    @MainActor @preconcurrency public
init(_ scale: Double, anchor: UnitPoint =
:center)

    /// Gets the current body of the
caller.
    ///
    /// `content` is a proxy for the view
that will have the modifier
    /// represented by `Self` applied to
it.
    @MainActor @preconcurrency public
func body(content:
ScaleTransition.Content, phase:
TransitionPhase) -> some View

    /// The type of view representing the
body.
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Body = some View
}

/// A dynamic property that scales a
numeric value.
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
```

```
@propertyWrapper public struct
ScaledMetric<Value> : DynamicProperty
where Value : BinaryFloatingPoint {

    /// Creates the scaled metric with an
    /// unscaled value and a text style to
    /// scale relative to.
    public init(wrappedValue: Value,
relativeTo textStyle: Font.TextStyle)

    /// Creates the scaled metric with an
    /// unscaled value using the default
    /// scaling.
    public init(wrappedValue: Value)

    /// The value scaled based on the
    /// current environment.
    public var wrappedValue: Value {
get }
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension ScaledMetric : Sendable where
Value : Sendable {

}

/// A shape with a scale transform
/// applied to it.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct
ScaledShape<Content> : Shape where
```

```
Content : Shape {  
  
    public var shape: Content  
  
    public var scale: CGSize  
  
    public var anchor: UnitPoint  
  
    @inlinable public init(shape:  
Content, scale: CGSize, anchor: UnitPoint  
= .center)  
  
        /// Describes this shape as a path  
        // within a rectangular frame of reference.  
        ///  
        /// - Parameter rect: The frame of  
        // reference for describing this shape.  
        ///  
        /// - Returns: A path that describes  
        // this shape.  
    nonisolated public func path(in rect:  
CGRect) -> Path  
  
        /// An indication of how to style a  
        // shape.  
        ///  
        /// SwiftUI looks at a shape's role  
        // when deciding how to apply a  
        /// ``ShapeStyle`` at render time.  
        The ``Shape`` protocol provides a  
        /// default implementation with a  
        // value of ``ShapeRole/fill``. If you  
        /// create a composite shape, you can
```

```
provide an override of this property
    /// to return another value, if
appropriate.
    @available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
    nonisolated public static var role:
ShapeRole { get }

    /// Returns the behavior this shape
should use for different layout
    /// directions.
    ///
    /// If the layoutDirectionBehavior
for a Shape is one that mirrors, the
    /// shape's path will be mirrored
horizontally when in the specified layout
    /// direction. When mirrored, the
individual points of the path will be
    /// transformed.
    ///
    /// Defaults to `mirrors` when
deploying on iOS 17.0, macOS 14.0,
    /// tvOS 17.0, watchOS 10.0 and
later, and to `fixed` if not.
    /// To mirror a path when deploying
to earlier releases, either use
    ///
`View.flipsForRightToLeftLayoutDirection`
for a filled or stroked
    /// shape or conditionally mirror the
points in the path of the shape.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
```

```
    nonisolated public var
layoutDirectionBehavior:
LayoutDirectionBehavior { get }

    /// The type defining the data to
animate.
    public typealias AnimatableData =
AnimatablePair<Content.AnimatableData,
AnimatablePair<CGSize.AnimatableData,
UnitPoint.AnimatableData>>

    /// The data to animate.
    public var animatableData:
ScaledShape<Content>.AnimatableData

    /// The type of view representing the
body of this view.
    ///
    /// When you create a custom view,
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Body
}

/// A type that defines the different
kinds of content offset adjusting
/// behaviors a scroll view can have.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct
```

```
ScrollContentOffsetAdjustmentBehavior {  
  
    /// The automatic behavior.  
    ///  
    /// A scroll view may automatically  
    adjust its content offset  
    /// based on the current context. The  
    absolute offset may be adjusted  
    /// to keep content in relatively the  
    same place. For example,  
    /// when scrolled to the bottom, a  
    scroll view may keep the bottom  
    /// edge scrolled to the bottom when  
    the overall size of its content  
    /// changes.  
    public static var automatic:  
    ScrollContentOffsetAdjustmentBehavior {  
        get }  
  
    /// The disabled behavior.  
    ///  
    /// A scroll view will not adjust its  
    content offset.  
    public static var disabled:  
    ScrollContentOffsetAdjustmentBehavior {  
        get }  
  
    /// A type that defines the geometry of a  
    scroll view.  
    ///  
    /// SwiftUI provides you values of this  
    type when using modifiers like
```

```
///  
``View/onScrollGeometryChange(_:action:)``  
` or  
/// ``View/onScrollPhaseChange(_:)``.  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
public struct ScrollGeometry : Equatable,  
Sendable {  
  
    /// The content offset of the scroll  
view.  
    ///  
    /// This is the position of the  
scroll view within its overall  
    /// content size. This value may  
extend before zero or beyond  
    /// the content size when the content  
insets of the scroll view  
    /// are non-zero or when rubber  
banding.  
    public var contentOffset: CGPoint  
  
    /// The size of the content of the  
scroll view.  
    ///  
    /// Unlike the container size of the  
scroll view, this refers to the  
    /// total size of the content of the  
scroll view which can be smaller  
    /// or larger than its containing  
size.  
    public var contentSize: CGSize
```

```
    /// The content insets of the scroll
view.
    /**
     /// Adding these insets to the
content size of the scroll view
     /// will give you the total
scrollable space of the scroll view.
public var contentInsets: EdgeInsets

    /// The size of the container of the
scroll view.
    /**
     /// This is the overall size of the
scroll view. Combining this
     /// and the content offset will give
you the current visible rect
     /// of the scroll view.
public var containerSize: CGSize

    /// The visible rect of the scroll
view.
    /**
     /// This value is computed from the
scroll view's content offset, content
     /// insets, and its container size.
public var visibleRect: CGRect {
get }

    /// The bounds rect of the scroll
view.
    /**
     /// Unlike the visible rect, this
value is within the content insets
```

```
    /// of the scroll view.  
    public var bounds: CGRect { get }  
  
    /// 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:  
ScrollGeometry, b: ScrollGeometry) ->  
Bool  
}
```

```
extension ScrollGeometry {  
  
    /// Creates a scroll geometry.  
    @available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
    public init(contentOffset: CGPoint,  
contentSize: CGSize, contentInsets:  
EdgeInsets, containerSize: CGSize)  
}
```

```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension ScrollGeometry :
```

```
CustomDebugStringConvertible {  
  
    /// A textual representation of this  
instance, suitable for debugging.  
    ///  
    /// Calling this property directly is  
discouraged. Instead, convert an  
    /// instance of any type to a string  
by using the `String(reflecting:)`  
    /// initializer. This initializer  
works with any type, and uses the custom  
    /// `debugDescription` property for  
types that conform to  
    /// `CustomDebugStringConvertible`:  
    ///  
    ///     struct Point:  
CustomDebugStringConvertible {  
    ///         let x: Int, y: Int  
    ///  
    ///         var debugDescription:  
String {  
    ///             return "(\(x), \(y))"  
    ///         }  
    ///     }  
    ///  
    ///     let p = Point(x: 21, y: 30)  
    ///     let s = String(reflecting: p)  
    ///     print(s)  
    ///     // Prints "(21, 30)"  
    ///  
    /// The conversion of `p` to a string  
in the assignment to `s` uses the  
    /// `Point` type's `debugDescription`
```

```
property.
```

```
    public var debugDescription: String {  
get }  
}
```

```
/// A type that describes the state of a  
scroll gesture of a
```

```
/// scrollable view like a scroll view.
```

```
///
```

```
/// A scroll gesture can be in one of  
four phases:
```

```
///     - idle: No active scroll is  
occurring.
```

```
///     - panning: An active scroll being  
driven by the user
```

```
///         is occurring.
```

```
///     - decelerating: The user has  
stopped driving a scroll
```

```
///         and the scroll view is  
decelerating to its final
```

```
///         target.
```

```
///     - animating: The system is  
animating to a final target
```

```
///         as a result of a programmatic  
animated scroll from
```

```
///             using a ``ScrollViewReader`` or
```

```
///
```

```
``View/scrollPosition(id:anchor:)``  
modifier.
```

```
///
```

```
/// SwiftUI provides you a value of this  
type when using the
```

```
/// ``View/onScrollPhaseChange(_:)``
```

modifier.

```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
@frozen public enum ScrollPhase :  
Equatable {  
  
    /// The idle phase where no kind of  
    /// scrolling is occurring.  
    case idle  
  
    /// The tracking phase where the  
    /// scroll view is tracking  
    /// a potential scroll by the user  
    /// but the user hasn't started  
    /// a scroll.  
    ///  
    /// For example, on iOS, the user may  
    /// start touching content  
    /// inside of the scroll view. Until  
    /// the user moves their finger  
    /// the scroll view would be tracking  
    /// the finger. Not all  
    /// platforms or kinds of scroll may  
    /// trigger this phase.  
    case tracking  
  
    /// The interacting phase where the  
    /// user is interacting  
    /// with the scroll view.  
    case interacting  
  
    /// The decelerating phase where the  
    /// user use has stopped
```

```
    /// interacting with the scroll view  
and the scroll view  
    /// is decelerating towards its final  
target.  
    case decelerating  
  
        /// The animating phase where the  
scroll view is animating  
        /// towards a final target.  
        ///  
        /// This phase is the result of a  
programmatic  
        /// scroll when using a  
``ScrollViewReader`` or  
        ///  
``View/scrollPosition(id:anchor:)``  
modifier.  
        ///  
        /// SwiftUI provides you a value of  
this type when using the  
        /// ``View/onScrollPhaseChange(_:)``  
modifier with a scrollable  
        /// view like ``ScrollView`` or  
``List``.  
    case animating  
  
        /// Whether the scroll view is  
actively scrolling.  
        ///  
        /// This convenience is equivalent to  
`phase != .idle`.  
    public var isScrolling: Bool { get }
```

```
    /// 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:  
        ScrollPhase, b: ScrollPhase) -> 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 }  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension ScrollPhase :  
CustomDebugStringConvertible {  
  
    /// A textual representation of this
```

instance, suitable for debugging.

```
///
/// Calling this property directly is
discouraged. Instead, convert an
/// instance of any type to a string
by using the `String(reflecting:)`
/// initializer. This initializer
works with any type, and uses the custom
/// `debugDescription` property for
types that conform to
/// `CustomDebugStringConvertible`:
///
///     struct Point:
CustomDebugStringConvertible {
    ///         let x: Int, y: Int
    ///
    ///         var debugDescription:
String {
    ///             return "(\(x), \(y))"
    ///         }
    ///     }
    ///
    ///     let p = Point(x: 21, y: 30)
    ///     let s = String(reflecting: p)
    ///     print(s)
    ///     // Prints "(21, 30)"
    ///
    /// The conversion of `p` to a string
in the assignment to `s` uses the
/// `Point` type's `debugDescription`
property.
    public var debugDescription: String {
get }
```

```
}
```

```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension ScrollPhase : Hashable {  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension ScrollPhase : Sendable {  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension ScrollPhase : BitwiseCopyable {  
}  
  
/// A type that defines the semantic  
position of where a scroll view is  
/// scrolled within its content.  
///  
/// Use this type along with the  
``View/scrollPosition(_:)``  
/// modifier to control where a scroll  
view is positioned. You  
/// can use this type to scroll in a  
variety of ways:  
/// - scroll to a view with a provided  
identity  
/// - scroll to a concrete offset  
/// - scroll to an edge  
///  
/// You can create a scroll position with
```

```
a specified view identity type
///
///      @State private var position =
ScrollPosition(idType: MyItem.ID.self)
///
/// SwiftUI will use that along with the
views in the scroll view's
/// scroll target layout to
programmatically scroll to those views
and to
/// update the ``ScrollPosition/viewID``
property as the user scrolls.
/// Use the ``View/scrollTargetLayout()``
modifier to configure
/// which layout contains your scroll
targets.
///
/// When scrolling to a view with an
identifier, SwiftUI will update
/// the position with the value of the
top-most view scrolled within
/// the visible region of the scroll
view.
///
/// In the following example, the
position binding will update to reflect
/// the top-most ItemView as the scroll
view scrolls.
///
///      @Binding var items: [MyItem]
///      @State private var position:
ScrollPosition
///          = .init(idType:
```

```
MyItem.ID.self)
////
////    ScrollView {
////        LazyVStack {
////            ForEach(items) { item in
////                ItemView(item)
////            }
////        }
////        .scrollTargetLayout()
////    }
////    .scrollPosition($scrolledID)
////
/// You can then query the currently
/// scrolled id by using the
/// ``ScrollPosition/viewID(type:)``.
////
///     let viewID: MyItem.ID =
/// position.viewID(type: MyItem.ID.self)
////
/// While most use cases will use view
/// identity based scrolling, you
/// can also use the scroll position type
/// to scroll to offsets or edges.
/// For example, you can create a button
/// that scrolls to the bottom of
/// the scroll view by specifying an
/// edge.
////
///     Button("Scroll to bottom") {
////
///         position.scrollTo(edge: .bottom)
///     }
///
```

```
/// When configuring a scroll position,  
SwiftUI will attempt to keep that  
/// position stable. For an edge, that  
means keeping a top aligned  
/// scroll view scrolled to the top if  
the content size changes.  
/// For a point, SwiftUI won't attempt to  
keep that exact offset scrolled  
/// when the content size changes nor  
will it update to a new offset  
/// when that changes.  
///  
/// For view identity positions, SwiftUI  
will attempt to keep the view with  
/// the identity specified in the  
provided binding visible when events  
occur  
/// that might cause it to be scrolled  
out of view by the system.  
/// Some examples of these include:  
/// - The data backing the content of a  
scroll view is re-ordered.  
/// - The size of the scroll view  
changes, like when a window is resized  
/// on macOS or during a rotation on  
iOS.  
/// - The scroll view initially lays  
out its content defaulting to  
/// the top most view, but the  
binding has a different view's identity.  
///  
/// You can provide an anchor to a view  
identity based position to:
```

```
/// - Influence which view the system
/// chooses as the view whose
/// identity value will update the
/// providing binding as the scroll
/// view scrolls.
/// - Control the alignment of the view
when scrolling to a view
/// when writing a new binding value.
///
/// In the example below, the bottom most
view will be chosen to update the
/// position binding with.
///
/// ScrollView {
///     LazyVStack {
///         ForEach(items) { item in
///             ItemView(item)
///         }
///     }
///     .scrollTargetLayout()
/// }
/// .scrollPosition($scrolledID,
anchor: .bottom)
///

/// For example, providing a value of
``UnitPoint/bottom`` will prefer
/// to have the bottom-most view chosen
and prefer to scroll to views
/// aligned to the bottom.
///

/// If no anchor has been provided,
SwiftUI will scroll the minimal amount
/// when using the scroll position to
```

```
programmatically scroll to a view.  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
public struct ScrollPosition : Sendable {  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension ScrollPosition {  
  
    /// Creates a new scroll position to  
    a view with a provided identity value.  
    ///  
    /// The type of the ID indicates the  
    type of ID of views within a  
    /// scroll target layout the scroll  
    view should look for.  
    @available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
    public init(id: some Hashable &  
Sendable, anchor: UnitPoint? = nil)  
  
    /// Creates a new automatic scroll  
    position.  
    ///  
    /// You can provide a type to the  
    scroll position. This type should match  
    /// the type of IDs associated to  
    views in a scroll target layout. The  
    /// scroll view will look for those  
    views to update the value of  
    /// the scroll position with.  
    @available(iOS 18.0, macOS 15.0, tvOS
```

```
18.0, watchOS 11.0, visionOS 2.0, *)
    public init(idType: (some Hashable &
Sendable).Type = Never.self)

        /// Creates a new scroll position to
        be scrolled to the provided edge.
        ///
        /// You can provide a type to
        indicate the type of ID the scroll view
        /// should look for views with an ID
        of that type within its scroll
        /// target layout.
        @available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
            public init(idType: (some Hashable &
Sendable).Type = Never.self, edge: Edge)

        /// Creates a new scroll position to
        be scrolled to the provided point.
        ///
        /// You can provide a type to the
        scroll position. This type should match
        /// the type of IDs associated to
        views in a scroll target layout. The
        /// scroll view will look for those
        views to update the value of
        /// the scroll position with.
        @available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
            public init(idType: (some Hashable &
Sendable).Type = Never.self, point:
CGPoint)
```

```
    /// Creates a new scroll position to
    // be scrolled to the provided x value.
    ///
    /// You can provide a type to the
    scroll position. This type should match
    /// the type of IDs associated to
    views in a scroll target layout. The
    /// scroll view will look for those
    views to update the value of
    /// the scroll position with.
    @available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public init(idType: (some Hashable &
Sendable).Type = Never.self, x: CGFloat,
y: CGFloat)

    /// Creates a new scroll position to
    // be scrolled to the provided y value.
    ///
    /// You can provide a type to the
    scroll position. This type should match
    /// the type of IDs associated to
    views in a scroll target layout. The
    /// scroll view will look for those
    views to update the value of
    /// the scroll position with.
    @available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public init(idType: (some Hashable &
Sendable).Type = Never.self, x: CGFloat)

    /// Creates a new scroll position to
    // be scrolled to the provided y value.
```

```
///  
/// You can provide a type to the  
scroll position. This type should match  
/// the type of IDs associated to  
views in a scroll target layout. The  
/// scroll view will look for those  
views to update the value of  
/// the scroll position with.  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
public init(idType: (some Hashable &  
Sendable).Type = Never.self, y: CGFloat)  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension ScrollPosition {  
  
    /// Scrolls the position of the  
scroll view to a view with a identity  
value  
    /// and anchor you provide.  
    ///  
    /// Inform the scroll view of which  
layout it should look for view's  
    /// with the identity value you  
provide using the  
    /// ``View/scrollTargetLayout()``  
modifier.  
    @available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
    public mutating func scrollTo(id:  
some Hashable & Sendable, anchor:
```

```
UnitPoint? = nil)

    /// Scrolls the position of the
    scroll view to the edge you provide.
    @available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public mutating func scrollTo(edge:
Edge)

    /// Scrolls the position of the
    scroll view to the point you provide.
    /**
     * The scroll view will clamp this
     * value to only scroll to the size of
     * its actual content.
     */
    @available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public mutating func scrollTo(point:
CGPoint)

    /// Scrolls the position of the
    scroll view to the x and y value you
    /**
     * provide.
     */
    /**
     * The scroll view will clamp this
     * value to only scroll to the size of
     * its actual content.
     */
    @available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    public mutating func scrollTo(x:
CGFloat, y: CGFloat)

    /// Scrolls the position of the
```

```
scroll view to the x value you provide.  
    ///  
    /// The scroll view chooses the y  
    value based on the content insets of  
    /// the scroll view and will clamp  
    this value to only scroll to the  
    /// size of its actual content.  
    @available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
    public mutating func scrollTo(x:  
CGFloat)  
  
        /// Scrolls the position of the  
        scroll view to the y value you provide.  
        ///  
        /// The scroll view chooses the x  
        value based on the content insets of  
        /// the scroll view and will clamp  
        this value to only scroll to the  
        /// size of its actual content.  
        @available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
        public mutating func scrollTo(y:  
CGFloat)  
    }  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension ScrollPosition {  
  
    /// Whether the scroll view has been  
    positioned by the user.  
    ///
```

```
    /// You can write to this property to
control whether the scroll view
    /// acts as if it has been positioned
by the user. If the position had
    /// a non-nil edge / point value,
that value will become nil when
    /// setting this property to true.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public var isPositionedByUser: Bool

    /// The positioned edge of the scroll
view if configured to be in that
    /// position.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public var edge: Edge? { get }

    /// The positioned point of the
scroll view if configured to be in that
    /// position.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public var point: CGPoint? { get }

    /// The type-erased id of the view
positioned in the scroll view if
    /// configured to be in that position
or the user has scrolled past a
    /// view with an id of matching type.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public var viewID: (any Hashable &
```

```
Sendable)? { get }
```

```
    /// The id of the view positioned in  
    /// the scroll view if configured  
    /// to be in that position or the  
    /// user has scrolled past a view with  
    /// an id of matching type.  
    @available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
    public func viewID<T>(type: T.Type)  
-> T? where T : Hashable, T : Sendable  
}
```

```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension ScrollPosition : Equatable {
```

```
    /// 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 == (lhs:  
        ScrollPosition, rhs: ScrollPosition) ->  
        Bool  
}
```

```
/// A type defining the target in which a
scroll view should try and scroll to.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct ScrollTarget {

    /// The rect that a scrollable view
    should try and have contained.
    public var rect: CGRect

    /// The anchor to which the rect
    should be aligned within the visible
    /// region of the scrollable view.
    public var anchor: UnitPoint?
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension ScrollTarget : Hashable,
Equatable {

    /// 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:  
ScrollTarget, b: ScrollTarget) -> 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 opaque collection representing the  
sections of view.
```

```
///
```

```
/// Sections are constructed lazily, on  
demand, so access only as much  
/// of this collection as is necessary to  
create the resulting content.
```

```
///
```

```
/// You can get access to a view's  
``SectionCollection`` by using the  
/// ``Group/init(sectionsOf:transform:)``  
initializer.
```

```
///
```

```
/// Any content of the given view which  
is not explicitly specified as a section  
/// is grouped with its sibling content  
to form implicit sections, meaning the
```

```
/// minimum number of sections in a
`SectionCollection` is one.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct SectionCollection : RandomAccessCollection {

    /// Accesses the element at the specified position.
    ///
    /// The following example accesses an element of an array through its
    /// subscript to print its value:
    ///
    ///     var streets = ["Adams",
    "Bryant", "Channing", "Douglas",
    "Evarts"]
    ///     print(streets[1])
    ///     // Prints "Bryant"
    ///
    /// You can subscript a collection with any valid index other than the
    /// collection's end index. The end index refers to the position one past
    /// the last element of a collection,
    so it doesn't correspond with an
    /// element.
    ///
    /// - Parameter position: The position of the element to access.
`position`
    /// must be a valid index of the collection that is not equal to the
```

```
    /// `endIndex` property.  
    ///  
    /// - Complexity: O(1)  
    public subscript(index: Int) ->  
SectionConfiguration { get }  
  
    /// The position of the first element  
in a nonempty collection.  
    ///  
    /// If the collection is empty,  
`startIndex` is equal to `endIndex`.  
    public var startIndex: Int { get }  
  
    /// The collection's "past the end"  
position---that is, the position one  
    /// greater than the last valid  
subscript argument.  
    ///  
    /// When you need a range that  
includes the last element of a  
collection, use  
    /// the half-open range operator  
(`..<`) with `endIndex`. The `..<`  
operator  
    /// creates a range that doesn't  
include the upper bound, so it's always  
    /// safe to use with `endIndex`. For  
example:  
    ///  
    ///     let numbers = [10, 20, 30,  
40, 50]  
    ///     if let index =  
numbers.firstIndex(of: 30) {
```

```
    ///         print(numbers[index ..<
numbers.endIndex])
    ///     }
    ///     // Prints "[30, 40, 50]"
    ///
    /// If the collection is empty,
`endIndex` is equal to `startIndex`.
public var endIndex: Int { get }

    /// A type representing the
sequence's elements.
@available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
public typealias Element =
SectionConfiguration

    /// A type that represents a position
in the collection.
///
/// Valid indices consist of the
position of every element and a
/// "past the end" position that's
not valid for use as a subscript
/// argument.
@available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
public typealias Index = Int

    /// A type that represents the
indices that are valid for subscripting
the
```

```
    /// collection, in ascending order.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias Indices = Range<Int>  
  
    /// A type that provides the  
collection's iteration interface and  
    /// encapsulates its iteration state.  
    ///  
    /// By default, a collection conforms  
to the `Sequence` protocol by  
    /// supplying `IndexingIterator` as  
its associated `Iterator`  
    /// type.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias Iterator =  
IndexingIterator<SectionCollection>  
  
    /// A collection representing a  
contiguous subrange of this collection's  
    /// elements. The subsequence shares  
indices with the original collection.  
    ///  
    /// The default subsequence type for  
collections that don't define their own  
    /// is `Slice`.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias SubSequence =
```

```
Slice<SectionCollection>
}

/// Specifies the contents of a section.
///
/// A `SectionConfiguration` includes the
content of the section, as well as
/// its header and footer.
///
/// A `SectionConfiguration` can
represent either an explicit section,
/// or groups of sibling views that are
not explicitly wrapped in a section.
///
/// Notably, the `header`, `footer` and
`content` properties of a
/// `SectionConfiguration` are all
`SubviewsCollection`s as they can be made
up
/// of multiple subviews. That means in
most cases, the subviews collection
/// should be treated as a collection
(either indexed into, or used with a
/// `ForEach`), or the subviews
collection should be wrapped in a
container
/// view, like a layout, or other custom
container:
///
///     PinboardSectionsLayout {
///         ForEach(sections: content)
{ section in
///             VStack {
```

```
/// HStack
{ section.header }
/// section.content
/// HStack
{ section.footer }
/// }
/// }
/// }
/// }

/// Here, we want to create one view for
`PinboardSectionsLayout` to place per
/// section in content. To do that, we
surround the `ForEach` body in another
/// container, a `VStack` layout,
ensuring the different subviews of
/// section.content are treated as a
single view by the surrounding layout.
/// Additionally, surrounding the header
and footer in an `HStack` layout
/// avoids vertically stacking subviews
of the header and footer
/// which we want visually grouped
together.

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct SectionConfiguration :  
Identifiable {  
  
    /// A unique identifier for a
section.  
    public struct ID : Hashable {  
  
        /// 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:  
SectionConfiguration.ID, b:  
SectionConfiguration.ID) -> 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 unique identifier representing
    the section.
    public var id:
SectionConfiguration.ID { get }

    /// The container values associated
    with the given section.
    ///
    /// Only explicitly created sections
    are able to have container values,
    /// meaning this container values
    will be empty if the section is implicit.
    public var containerValues:
ContainerValues { get }

    /// The contents of the section
    header.
    ///
    /// Notably, the section's header is
    a `SubviewsCollection`, not a
    /// `Subview`, as it can be made up
    of multiple subviews. That means in most
    /// cases, the subviews collection
    should be treated as a collection (either
    /// indexed into, or used with a
    `ForEach`), or the subviews collection
    /// should be wrapped in a container
    view, like a layout, or other custom
    /// container:
    ///
    ///     ForEach(sections: content) {
    ///         VStack {
    ///             HStack
```

```
{ section.header }
    /**
     * HStack
{ section.footer }
    /**
     */
    /**
     */
    /**
     * Here, we surround the header and
     * footer in an `HStack` layout to avoid
     * vertically stacking the subviews
     * of the header and footer which we want
     * visually grouped together.
Additionally, we surround the `ForEach`
body
    /**
     * in a VStack, so it is treated as
     * a single view by containers it gets
     * passed to.
public var header: SubviewsCollection
{ get }

    /**
     * The contents of the section
     * footer.
    /**
     * Notably, the section's footer is
     * a `SubviewsCollection`, not a
     * `Subview`, as it can be made up
     * of multiple subviews. That means in most
     * cases, the subviews collection
     * should be treated as a collection (either
     * indexed into, or used with a
     * `ForEach`), or the subviews collection
     * should be wrapped in a container
     * view, like a layout, or other custom
     * container:
```

```
///  
///     ForEach(sections: content) {  
///         VStack {  
///             HStack  
{ section.header }  
///             HStack  
{ section.footer }  
///         }  
///  
///         /// Here, we surround the header and  
///         footer in an `HStack` layout to avoid  
///         /// vertically stacking the subviews  
///         of the header and footer which we want  
///         /// visually grouped together.  
Additionally, we surround the `ForEach`  
body  
    /// in a VStack, so it is treated as  
a single view by containers it gets  
    /// passed to.  
    public var footer: SubviewsCollection  
{ get }  
  
    /// The contents of the section body.  
    ///  
    /// Notably, the section's content is  
a `SubviewsCollection`, not a  
    /// `Subview`, as it can be made up  
of multiple subviews. That means in most  
    /// cases, the subviews collection  
should be treated as a collection (either  
    /// indexed into, or used with a  
`ForEach`), or the subviews collection
```

```
    /// should be wrapped in a container
view, like a layout, or other custom
    /// container:
    ///
    ///     PinboardSectionsLayout {
    ///         ForEach(sections:
content) { section in
    ///
    ///             VStack {
    ///                 section.content
    ///             }
    ///
    ///         }
    ///
    /// Here, we want to create one view
for `PinboardSectionsLayout` to place
    /// per section in content. To do
that, we surround the `ForEach` body in
    /// another container, a `VStack`
layout, ensuring the different subviews
of
    /// section.content are treated as a
single view by the surrounding layout.
    public var content:
SubviewsCollection { get }
}

/// A style appropriate for foreground
separator or border lines.
///
/// You can also use
``ShapeStyle/separator`` to construct
this style.
@available(iOS 17.0, macOS 10.15, tvOS
```

```
17.0, watchOS 10.0, *)
public struct SeparatorShapeStyle : ShapeStyle {

    /// Creates a new separator shape
    style instance.
    public init()

    /// The type of shape style this will
    resolve to.
    ///
    /// When you create a custom shape
    style, Swift infers this type
    /// from your implementation of the
    required `resolve` function.
    @available(iOS 17.0, tvOS 17.0,
    watchOS 10.0, macOS 14.0, *)
    public typealias Resolved = Never
}

/// A reference to a function in a Metal
/// shader library, along with its
/// bound uniform argument values.
///
/// Shader values can be used as filter
/// effects on views, see the
/// ``View/colorEffect(_:_:)``,
///
/// ``View/distortionEffect(_:_:)``,
/// and
/// ``View/layerEffect(_:_:)`` functions.
```

```
///  
/// Shaders also conform to the  
``ShapeStyle`` protocol, letting their  
/// MSL shader function provide per-pixel  
color to fill any shape or  
/// text view. For a shader function to  
act as a fill pattern it must  
/// have a function signature matching:  
///  
///     [[ stitchable ]] half4  
name(float2 position, args...)  
///  
/// where `position` is the user-space  
coordinates of the pixel applied  
/// to the shader, and `args...` should  
be compatible with the uniform  
/// arguments bound to `shader`. The  
function should return the  
/// premultiplied color value in the  
color space of the destination  
/// (typically extended sRGB).  
///  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, *)  
@available(watchOS, unavailable)  
public struct Shader : Equatable,  
Sendable {  
  
    /// A single uniform argument value  
    // to a shader function.  
    public struct Argument : Equatable,  
Sendable {
```

```
        /// Returns an argument value
        representing the MSL value
        /// `float(x)`.

        public static func float<T>(_ x:
T) -> Shader.Argument where T :
BinaryFloatingPoint

        /// Returns an argument value
        representing the MSL value
        /// `float2(x, y)`.

        public static func float2<T>(_ x:
T, _ y: T) -> Shader.Argument where T :
BinaryFloatingPoint

        /// Returns an argument value
        representing the MSL value
        /// `float3(x, y, z)`.

        public static func float3<T>(_ x:
T, _ y: T, _ z: T) -> Shader.Argument
where T : BinaryFloatingPoint

        /// Returns an argument value
        representing the MSL value
        /// `float4(x, y, z, w)`.

        public static func float4<T>(_ x:
T, _ y: T, _ z: T, _ w: T) ->
Shader.Argument where T :
BinaryFloatingPoint

        /// Returns an argument value
        representing the MSL value
        /// `float2(point.x, point.y)`.

        public static func float2(_
```

```
point: CGPoint) -> Shader.Argument

    /// Returns an argument value
representing the MSL value
    /// `float2(size.width,
size.height)`.

public static func float2(_ size:
CGSize) -> Shader.Argument

    /// Returns an argument value
representing the MSL value
    /// `float2(vector.dx,
vector.dy)`.

public static func float2(_
vector: CGVector) -> Shader.Argument

    /// Returns an argument value
defined by the provided array of
    /// floating point numbers. When
passed to an MSL function it
    /// will convert to a `device
const float *ptr, int count` pair
    /// of parameters.

public static func floatArray(_
array: [Float]) -> Shader.Argument

    /// Returns an argument value
representing the bounding rect of
    /// the shape or view that the
shader is attached to, as
    /// `float4(x, y, width,
height)`. This value is undefined for
    /// shaders that do not have a
```

```
natural bounding rect (e.g.  
    /// filter effects drawn into  
`GraphicsContext`).  
    public static var boundingRect:  
Shader.Argument { get }  
  
        /// Returns an argument value  
representing `color`. When passed  
        /// to a MSL function it will  
convert to a `half4` value, as a  
        /// premultiplied color in the  
target color space.  
    public static func color(_ color:  
Color) -> Shader.Argument  
  
        /// Returns an argument value  
defined by the provided array of  
        /// color values. When passed to  
an MSL function it will convert  
        /// to a `device const half4  
*ptr, int count` pair of  
        /// parameters.  
    public static func colorArray(_  
array: [Color]) -> Shader.Argument  
  
        /// Returns an argument value  
defined by the provided image.  
        /// When passed to an MSL  
function it will convert to a  
        /// `texture2d<half>` value.  
Currently only one image parameter  
        /// is supported per `Shader`  
instance.
```

```
    public static func image(_ image:  
Image) -> Shader.Argument
```

```
        /// Returns an argument value  
defined by the provided data  
        /// value. When passed to an MSL  
function it will convert to a  
        /// `device const void *ptr, int  
size_in_bytes` pair of  
        /// parameters.  
    public static func data(_ data:  
Data) -> Shader.Argument
```

```
        /// 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:  
Shader.Argument, b: Shader.Argument) ->  
Bool  
}
```

```
    /// The shader function called by the  
shader.  
    public var function: ShaderFunction
```

```
    /// The uniform argument values  
    passed to the shader function.
```

```
    public var arguments:  
        [Shader.Argument]
```

```
    /// For shader functions that return  
    color values, whether the
```

```
        /// returned color has dither noise  
        added to it, or is simply
```

```
        /// rounded to the output bit-depth.
```

```
For shaders generating smooth
```

```
        /// gradients, dithering is usually  
        necessary to prevent visible
```

```
        /// banding in the result.
```

```
    public var dithersColor: Bool
```

```
    /// Creates a new shader from a  
    function and the uniform argument
```

```
        /// values to bind to the function.
```

```
    public init(function: ShaderFunction,  
    arguments: [Shader.Argument])
```

```
    /// 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: Shader, b:
    Shader) -> Bool
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, visionOS 2.0, *)
@available(watchOS, unavailable)
extension Shader {

    /// Attempts to asynchronously
    compile a shader function, to
    /// minimize the chance of stalling
    when it is first used for
    /// rendering.
    ///
    /// Before being available for
    rendering each shader must be
    /// compiled for the current GPU. By
    default this happens on first
    /// use, but that may cause that
    frame to be delayed waiting for
    /// the compiler, i.e. cause a
    visible "glitch" in animations.
    ///
    /// Calling this method for each
    shader before it's first used
    /// allows the necessary compilation
    to happen ahead of time,
    /// eliminating the delay on the
    first actual use (provided
    /// `compile()` completes before the
```

first use of the shader).

```
///
/// For compilation to be successful
the specified usage type must
/// match how the shader is
eventually used to render, and its
/// current argument values must
match the types of the arguments
/// used when rendering (however
array and data sizes may be
/// empty).
///
/// For example, to compile a fill
shader asynchronously when your
/// app launches:
///
/// Task {
///     let shader =
ShaderLibrary.example(.color(.clear), .fl
oat(0))
///         try! await
shader.compile(as: .shapeStyle)
///     }
///
/// Here the MSL shader function
`example` takes two uniform
/// arguments: a color and a numeric
value. The placeholder values
/// are replaced with actual values
when using the shader, in this
/// case to fill a circle:
///
///     Circle().fill()
```

```
    /**
     * - Parameter type: how the shader will eventually be used.
     */
    /**
     * - Throws: an error describing why compilation failed.
     */
    public func compile(as type: Shader.UsageType) async throws

        /**
         * The different ways in which a `Shader` may be used to render.
         */
        public struct UsageType : Hashable, Sendable {

            /**
             * The shader will be used as a `ShapeStyle` value.
             */
            public static let shapeStyle: Shader.UsageType

            /**
             * The shader will be used as a color effect.
             */
            public static let colorEffect: Shader.UsageType

            /**
             * The shader will be used as a geometric distortion effect.
             */
            public static let distortionEffect: Shader.UsageType

            /**
             * The shader will be used as a
```

layer effect.

```
public static let layerEffect:  
Shader.UsageType
```

```
    /// 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:  
Shader.UsageType, b: Shader.UsageType) ->  
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 }
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, *)
@available(watchOS, unavailable)
extension Shader {

    /// The type of shape style this will
    resolve to.
    ///
    /// When you create a custom shape
    style, Swift infers this type
    /// from your implementation of the
    required `resolve` function.
    @available(iOS 17.0, tvOS 17.0,
    watchOS 10.0, macOS 14.0, *)
    public typealias Resolved = Never
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, *)
@available(watchOS, unavailable)
extension Shader : ShapeStyle {
}

/// A reference to a function in a Metal
// shader library.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, *)
```

```
@available(watchOS, unavailable)
@dynamicCallable public struct
ShaderFunction : Equatable, Sendable {

    /// The shader library storing the
    function.
    public var library: ShaderLibrary

    /// The name of the shader function
    in the library.
    public var name: String

    /// Creates a new function reference
    from the provided shader
    /// library and function name string.
    public init(library: ShaderLibrary,
name: String)

    /// Returns a new shader by applying
    the provided argument values
    /// to the referenced function.
    ///
    /// Typically this subscript is used
    implicitly via function-call
    /// syntax, for example:
    ///
    ///     let shader =
    ShaderLibrary.default.myFunction(.float(4
2))
    ///
    /// which creates a shader passing
    the value `42` to the first
    /// unbound parameter of
```

```
`myFunction()`.  
    public func  
dynamicallyCall(withArguments args:  
[Shader.Argument]) -> Shader  
  
    /// 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:  
ShaderFunction, b: ShaderFunction) ->  
Bool  
}  
  
/// A Metal shader library.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, *)  
@available(watchOS, unavailable)  
@dynamicMemberLookup public struct  
ShaderLibrary : Equatable, @unchecked  
Sendable {  
  
    /// The default shader library of the  
main (i.e. app) bundle.  
    public static let `default`:
```

## ShaderLibrary

```
    /// Returns the default shader
    library of the specified bundle.
    public static func bundle(_ bundle:
Bundle) -> ShaderLibrary

    /// Creates a new Metal shader
    library from `data`, which must be
    /// the contents of precompiled Metal
    library. Functions compiled
    /// from the returned library will
    only be cached as long as the
    /// returned library exists.
    public init(data: Data)

    /// Creates a new Metal shader
    library from the contents of `url`,
    /// which must be the location of
    precompiled Metal library.
    /// Functions compiled from the
    returned library will only be
    /// cached as long as the returned
    library exists.
    public init(url: URL)

    /// Returns a new shader function
    representing the stitchable MSL
    /// function called `name` in the
    default shader library.
    ///
    /// Typically this subscript is used
    implicitly via the dynamic
```

```
    /// member syntax, for example:  
    ///  
    ///     let fn =  
ShaderLibrary.myFunction  
    ///  
    /// which creates a reference to the  
MSL function called  
    /// `myFunction()`.  
    public static subscript(dynamicMember  
name: String) -> ShaderFunction { get }  
  
    /// Returns a new shader function  
representing the stitchable MSL  
    /// function in the library called  
`name`.  
    ///  
    /// Typically this subscript is used  
implicitly via the dynamic  
    /// member syntax, for example:  
    ///  
    ///     let fn =  
ShaderLibrary.default.myFunction  
    ///  
    /// which creates a reference to the  
MSL function called  
    /// `myFunction()`.  
    public subscript(dynamicMember name:  
String) -> ShaderFunction { get }  
  
    /// 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 == (lhs:  
ShaderLibrary, rhs: ShaderLibrary) ->  
Bool  
}
```

```
/// A style to use when rendering  
shadows.
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
public struct ShadowStyle : Equatable,  
Sendable {
```

```
    /// Creates a custom drop shadow  
style.
```

```
    ///  
    /// Drop shadows draw behind the  
source content by blurring,  
    /// tinting and offsetting its per-  
pixel alpha values.
```

```
    ///  
    /// - Parameters:  
    ///     - color: The shadow's color.  
    ///     - radius: The shadow's size.  
    ///     - x: A horizontal offset you  
use to position the shadow
```

```
    ///      relative to this view.  
    /// - y: A vertical offset you use  
to position the shadow  
    ///      relative to this view.  
    ///  
    /// - Returns: A new shadow style.  
public static func drop(color: Color  
= .init(.sRGBLinear, white: 0, opacity:  
0.33), radius: CGFloat, x: CGFloat = 0,  
y: CGFloat = 0) -> ShadowStyle  
  
    /// Creates a custom inner shadow  
style.  
    ///  
    /// Inner shadows draw on top of the  
source content by blurring,  
    /// tinting, inverting and offsetting  
its per-pixel alpha values.  
    ///  
    /// - Parameters:  
    /// - color: The shadow's color.  
    /// - radius: The shadow's size.  
    /// - x: A horizontal offset you  
use to position the shadow  
    ///      relative to this view.  
    /// - y: A vertical offset you use  
to position the shadow  
    ///      relative to this view.  
    ///  
    /// - Returns: A new shadow style.  
public static func inner(color: Color  
= .init(.sRGBLinear, white: 0, opacity:  
0.55), radius: CGFloat, x: CGFloat = 0,
```

```
y: CGFloat = 0) -> ShadowStyle

    /// 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:
ShadowStyle, b: ShadowStyle) -> Bool
}
```

```
/// A 2D shape that you can use when
drawing a view.
///
/// Shapes without an explicit fill or
stroke get a default fill based on the
/// foreground color.
///
/// You can define shapes in relation to
an implicit frame of reference, such as
/// the natural size of the view that
contains it. Alternatively, you can
define
/// shapes in terms of absolute
coordinates.
```

```
@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
public protocol Shape : Sendable,
Animatable, View,
_RemoveGlobalActorIsolation {

    /// Describes this shape as a path
    within a rectangular frame of reference.
    ///
    /// - Parameter rect: The frame of
    reference for describing this shape.
    ///
    /// - Returns: A path that describes
    this shape.
    nonisolated func path(in rect:
    CGRect) -> Path

    /// An indication of how to style a
    shape.
    ///
    /// SwiftUI looks at a shape's role
    when deciding how to apply a
    /// ``ShapeStyle`` at render time.
    The ``Shape`` protocol provides a
    /// default implementation with a
    value of ``ShapeRole/fill``. If you
    /// create a composite shape, you can
    provide an override of this property
    /// to return another value, if
    appropriate.
    @available(iOS 15.0, macOS 12.0, tvOS
    15.0, watchOS 8.0, *)
    nonisolated static var role:
    ShapeRole { get }
```

```
    /// Returns the behavior this shape  
    should use for different layout  
    /// directions.  
    ///  
    /// If the layoutDirectionBehavior  
    for a Shape is one that mirrors, the  
    /// shape's path will be mirrored  
    horizontally when in the specified layout  
    /// direction. When mirrored, the  
    individual points of the path will be  
    /// transformed.  
    ///  
    /// Defaults to `mirrors` when  
    deploying on iOS 17.0, macOS 14.0,  
    /// tvOS 17.0, watchOS 10.0 and  
    later, and to `fixed` if not.  
    /// To mirror a path when deploying  
    to earlier releases, either use  
    ///  
    `View.flipsForRightToLeftLayoutDirection`  
    for a filled or stroked  
    /// shape or conditionally mirror the  
    points in the path of the shape.  
    @available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
    nonisolated var  
layoutDirectionBehavior:  
LayoutDirectionBehavior { get }  
  
    /// Returns the size of the view that  
    will render the shape, given  
    /// a proposed size.
```

```
///  
/// Implement this method to tell the  
container of the shape how  
/// much space the shape needs to  
render itself, given a size  
/// proposal.  
///  
/// See  
``Layout/sizeThatFits(proposal:subviews:c  
ache:)``  
/// for more details about how the  
layout system chooses the size of  
/// views.  
///  
/// - Parameters:  
///   - proposal: A size proposal for  
the container.  
///  
/// - Returns: A size that indicates  
how much space the shape needs.  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
nonisolated func sizeThatFits(_  
proposal: ProposedViewSize) -> CGSize  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Shape {  
  
    /// Trims this shape by a fractional  
amount based on its representation as a  
/// path.
```

```
///
/// To create a `Shape` instance, you
define the shape's path using lines and
/// curves. Use the `trim(from:to:)` method to draw a portion of a shape by
/// ignoring portions of the beginning and ending of the shape's path.
///
/// For example, if you're drawing a figure eight or infinity symbol ( $\infty$ )
/// starting from its center, setting the `startFraction` and `endFraction`
/// to different values determines the parts of the overall shape.
///
/// The following example shows a simplified infinity symbol that draws
/// only three quarters of the full shape. That is, of the two lobes of the
/// symbol, one lobe is complete and the other is half complete.
///
/// Path { path in
    path.addLines([
        .init(x: 2, y: 1),
        .init(x: 1, y: 0),
        .init(x: 0, y: 1),
        .init(x: 1, y: 2),
        .init(x: 3, y: 0),
        .init(x: 4, y: 1),
        .init(x: 3, y: 2),
        .init(x: 2, y: 1)
    ])
}
```

```
        /**
         * .trim(from: 0.25, to: 1.0)
         * .scale(50,
anchor: .topLeading)
        *.stroke(Color.black,
lineWidth: 3)
        /**
         /// Changing the parameters of
`trim(from:to:)` to
        /// `.trim(from: 0, to: 1)` draws the
full infinity symbol, while
        /// `.trim(from: 0, to: 0.5)` draws
only the left lobe of the symbol.
        /**
         /// - Parameters:
         ///   - startFraction: The fraction
of the way through drawing this shape
         ///     where drawing starts.
         ///   - endFraction: The fraction of
the way through drawing this shape
         ///     where drawing ends.
         /// - Returns: A shape built by
capturing a portion of this shape's path.
    @inlinable nonisolated public func
trim(from startFraction: CGFloat = 0, to
endFraction: CGFloat = 1) -> some Shape

}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension Shape {
```

```
    /// Returns the original proposal,  
    with nil components replaced by  
    /// a small positive value.  
    nonisolated public func  
sizeThatFits(_ proposal:  
ProposedViewSize) -> CGSize  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Shape {  
  
    /// An indication of how to style a  
shape.  
    ///  
    /// SwiftUI looks at a shape's role  
when deciding how to apply a  
    /// ``ShapeStyle`` at render time.  
The ``Shape`` protocol provides a  
    /// default implementation with a  
value of ``ShapeRole/fill``. If you  
    /// create a composite shape, you can  
provide an override of this property  
    /// to return another value, if  
appropriate.  
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    public static var role: ShapeRole {  
get }  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)
```

```
extension Shape {  
  
    /// Returns the behavior this shape  
    /// should use for different layout  
    /// directions.  
    ///  
    /// If the layoutDirectionBehavior  
    /// for a Shape is one that mirrors, the  
    /// shape's path will be mirrored  
    /// horizontally when in the specified layout  
    /// direction. When mirrored, the  
    /// individual points of the path will be  
    /// transformed.  
    ///  
    /// Defaults to `mirrors` when  
    /// deploying on iOS 17.0, macOS 14.0,  
    /// tvOS 17.0, watchOS 10.0 and  
    /// later, and to `fixed` if not.  
    /// To mirror a path when deploying  
    /// to earlier releases, either use  
    ///  
    `View.flipsForRightToLeftLayoutDirection`  
    /// for a filled or stroked  
    /// shape or conditionally mirror the  
    /// points in the path of the shape.  
    public var layoutDirectionBehavior:  
        LayoutDirectionBehavior { get }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Shape {
```

```
    /// Returns a new shape that is a
    stroked copy of `self`, using the
    /// contents of `style` to define the
    stroke characteristics.
    @inlinable nonisolated public func
stroke(style: StrokeStyle) -> some Shape
```

```
    /// Returns a new shape that is a
    stroked copy of `self` with
    /// line-width defined by `lineWidth`
    and all other properties of
    /// `StrokeStyle` having their
    default values.
```

```
    @inlinable nonisolated public func
stroke(lineWidth: CGFloat = 1) -> some
Shape
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Shape where Self == Rectangle {
```

```
    /// A rectangular shape aligned
    inside the frame of the view containing
    it.
```

```
    public static var rect: Rectangle {
get }
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
```

```
extension Shape where Self ==  
RoundedRectangle {  
  
    /// A rectangular shape with rounded  
corners, aligned inside the frame of  
    /// the view containing it.  
    public static func rect(cornerSize:  
CGSize, style: RoundedCornerStyle  
= .continuous) -> Self  
  
    /// A rectangular shape with rounded  
corners, aligned inside the frame of  
    /// the view containing it.  
    public static func rect(cornerRadius:  
CGFloat, style: RoundedCornerStyle  
= .continuous) -> Self  
}  
  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension Shape where Self ==  
UnevenRoundedRectangle {  
  
    /// A rectangular shape with rounded  
corners with different values, aligned  
    /// inside the frame of the view  
containing it.  
    public static func rect(cornerRadii:  
RectangleCornerRadii, style:  
RoundedCornerStyle = .continuous) -> Self  
  
    /// A rectangular shape with rounded  
corners with different values, aligned
```

```
    /// inside the frame of the view
    containing it.
    public static func
rect(topLeadingRadius: CGFloat = 0,
bottomLeadingRadius: CGFloat = 0,
bottomTrailingRadius: CGFloat = 0,
topTrailingRadius: CGFloat = 0, style:
RoundedCornerStyle = .continuous) -> Self
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Shape where Self == Capsule {

    /// A capsule shape aligned inside
    the frame of the view containing it.
    ///
    /// A capsule shape is equivalent to
    a rounded rectangle where the corner
    /// radius is chosen as half the
    length of the rectangle's smallest edge.
    public static var capsule: Capsule {
get }

    /// A capsule shape aligned inside
    the frame of the view containing it.
    ///
    /// A capsule shape is equivalent to
    a rounded rectangle where the corner
    /// radius is chosen as half the
    length of the rectangle's smallest edge.
    public static func capsule(style:
RoundedCornerStyle) -> Self
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Shape where Self == Ellipse {
```

```
    /// An ellipse aligned inside the  
    frame of the view containing it.  
    public static var ellipse: Ellipse {  
        get }  
    }
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Shape where Self == Circle {  
  
    /// A circle centered on the frame of  
    the view containing it.  
    ///  
    /// The circle's radius equals half  
    the length of the frame rectangle's  
    /// smallest edge.  
    public static var circle: Circle {  
        get }  
    }
```

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

```
    /// Returns a new shape with filled  
    regions common to both shapes.  
    ///
```

```
    /// - Parameters:  
    ///   - other: The shape to  
    intersect.  
    ///   - eoFill: Whether to use the  
    even-odd rule for determining  
    ///           which areas to treat as the  
    interior of the shapes (if true),  
    ///           or the non-zero rule (if  
    false).  
    /// - Returns: A new shape.  
    ///  
    /// The filled region of the  
    resulting shape is the overlapping area  
    /// of the filled region of both  
    shapes. This can be used to clip  
    /// the fill of a shape to a mask.  
    ///  
    /// Any unclosed subpaths in either  
    shape are assumed to be closed.  
    /// The result of filling this shape  
    using either even-odd or  
    /// non-zero fill rules is identical.  
    nonisolated public func  
intersection<T>(_ other: T, eoFill: Bool  
= false) -> some Shape where T : Shape
```

```
    /// Returns a new shape with filled  
    regions in either this shape or  
    /// the given shape.  
    ///  
    /// - Parameters:  
    ///   - other: The shape to union.
```

```
    /// - eoFill: Whether to use the
even-odd rule for determining
    /// which areas to treat as the
interior of the shapes (if true),
    /// or the non-zero rule (if
false).
    /// - Returns: A new shape.
    ///
    /// The filled region of resulting
shape is the combination of the
    /// filled region of both shapes
added together.
    ///
    /// Any unclosed subpaths in either
shape are assumed to be closed.
    /// The result of filling this shape
using either even-odd or
    /// non-zero fill rules is identical.
nonisolated public func union<T>(_
other: T, eoFill: Bool = false) -> some
Shape where T : Shape
```

```
    /// Returns a new shape with filled
regions from this shape that are
    /// not in the given shape.
    ///
    /// - Parameters:
    ///   - other: The shape to subtract.
    ///   - eoFill: Whether to use the
even-odd rule for determining
    /// which areas to treat as the
interior of the shapes (if true),
```

```
    /// or the non-zero rule (if
false).
    /// - Returns: A new shape.
    ///
    /// The filled region of the
resulting shape is the filled region of
    /// this shape with the filled
region `other` removed from it.
    ///
    /// Any unclosed subpaths in either
shape are assumed to be closed.
    /// The result of filling this shape
using either even-odd or
    /// non-zero fill rules is identical.
nonisolated public func
subtracting<T>(_ other: T, eoFill: Bool =
false) -> some Shape where T : Shape
```

```
    /// Returns a new shape with filled
regions either from this shape or
    /// the given shape, but not in both.
    ///
    /// - Parameters:
    ///   - other: The shape to
difference.
    ///   - eoFill: Whether to use the
even-odd rule for determining
    ///           which areas to treat as the
interior of the shapes (if true),
    ///           or the non-zero rule (if
false).
    /// - Returns: A new shape.
```

```
    /**
     * The filled region of the
     * resulting shape is the filled region
     * contained in either this shape or
     * `other`, but not both.
     */
    /**
     * Any unclosed subpaths in either
     * shape are assumed to be closed.
     */
    /**
     * The result of filling this shape
     * using either even-odd or
     * non-zero fill rules is identical.
     */
    nonisolated public func
symmetricDifference<T>(_ other: T,
eoFill: Bool = false) -> some Shape where
T : Shape
```

```
    /**
     * Returns a new shape with a line
     * from this shape that overlaps the
     * filled regions of the given
     * shape.
     */
    /**
     * - Parameters:
     *   - other: The shape to
     *     intersect.
     */
    /**
     *   - eoFill: Whether to use the
     *     even-odd rule for determining
     *       which areas to treat as the
     *     interior of the shapes (if true),
     *       or the non-zero rule (if
     *     false).
     */
    /**
     *   - Returns: A new shape.
     */
```

```
    /// The line of the resulting shape  
    // is the line of this shape that  
    /// overlaps the filled region of  
`other`.  
    ///  
    /// Intersected subpaths that are  
clipped create open subpaths.  
    /// Closed subpaths that do not  
intersect `other` remain closed.  
    nonisolated public func  
lineIntersection<T>(_ other: T, eoFill:  
Bool = false) -> some Shape where T :  
Shape
```

```
    /// Returns a new shape with a line  
from this shape that does not  
    /// overlap the filled region of the  
given shape.  
    ///  
    /// – Parameters:  
    ///     – other: The shape to subtract.  
    ///     – eoFill: Whether to use the  
even–odd rule for determining  
    ///             which areas to treat as the  
interior of the shapes (if true),  
    ///             or the non–zero rule (if  
false).  
    /// – Returns: A new shape.  
    ///  
    /// The line of the resulting shape  
is the line of this shape that  
    /// does not overlap the filled
```

```
region of `other`.  
    ///  
    /// Intersected subpaths that are  
    clipped create open subpaths.  
    /// Closed subpaths that do not  
    intersect `other` remain closed.  
    nonisolated public func  
lineSubtraction<T>(_ other: T, eoFill:  
Bool = false) -> some Shape where T :  
Shape  
  
}  
  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
extension Shape where Self ==  
ContainerRelativeShape {  
  
    /// A shape that is replaced by an  
    inset version of the current  
    /// container shape. If no container  
    shape was defined, is replaced by  
    /// a rectangle.  
    public static var containerRelative:  
ContainerRelativeShape { get }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Shape {  
  
    /// Returns a new version of self  
    representing the same shape, but
```

```
    /// that will ask it to create its
    path from a rect of `size`. This
    /// does not affect the layout
    properties of any views created from
    /// the shape (e.g. by filling it).
    @inlinable nonisolated public func
size(_ size: CGSize) -> some Shape

    /// Returns a new version of self
    representing the same shape, but
    /// that will ask it to create its
    path from a rect of size
    /// `(width, height)`. This does not
    affect the layout properties
    /// of any views created from the
    shape (e.g. by filling it).
    @inlinable nonisolated public func
size(width: CGFloat, height: CGFloat) ->
some Shape

}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Shape {

    /// Returns a new version of self
    representing the same shape, but
    /// within a rect of `size` instead
    of the container size.
    ///
    /// The `anchor` parameter determines
```

```
how the shape will be positioned
    /// within the container when the
path's bounds and the container's bounds
    /// are not equal. This does not
affect the layout properties of any views
    /// created from the shape (e.g. by
filling it).
    ///
    /// - Parameters:
    ///   - size: The size to constrain
the shape to.
    ///   - anchor: The anchor to use to
determine how to position the new
    ///   shape.
    /// - Returns: A new shape
constrained to the given `size`, and
positioned
    /// using `anchor`.
nonisolated public func size(_ size:
CGSize, anchor: UnitPoint) -> some Shape
```

```
    /// Returns a new version of self
representing the same shape, but
    /// within a rect of `(width,
height)` instead of the container size.
    ///
    /// The `anchor` parameter determines
how the shape will be positioned
    /// within the container when the
path's bounds and the container's bounds
    /// are not equal. This does not
affect the layout properties of any views
```

```
    /// created from the shape (e.g. by
filling it).
    /**
     * - Parameters:
     *   - width: The width to constrain
the shape to.
     *   - height: The height to
constrain the shape to.
     *   - anchor: The anchor to use to
determine how to position the new
     *   shape.
     * - Returns: A new shape
constrained to the given `size`, and
positioned
     * using `anchor`.
nonisolated public func size(width:
CGFloat, height: CGFloat, anchor:
UnitPoint) -> some Shape
```

}

```
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Shape {
```

```
    /// Changes the relative position of
this shape using the specified size.
    /**
     * The following example renders two
circles. It places one circle at its
     * default position. The second
circle is outlined with a stroke,
     * positioned on top of the first
```

```
circle and offset by 100 points to the
    /// left and 50 points below.
    ///
    ///     Circle()
    ///     .overlay(
    ///         Circle()
    ///         .offset(CGSize(width:
-100, height: 50))
    ///             .stroke()
    ///         )
    ///
    /// - Parameter offset: The amount,
in points, by which you offset the
    /// shape. Negative numbers are to
the left and up; positive numbers are
    /// to the right and down.
    ///
    /// - Returns: A shape offset by the
specified amount.

    @inlinable public func offset(_
offset: CGSize) -> OffsetShape<Self>
```

```
    /// Changes the relative position of  
    this shape using the specified point.  
    ///  
    /// The following example renders two  
    circles. It places one circle at its  
    /// default position. The second  
    circle is outlined with a stroke,  
    /// positioned on top of the first  
    circle and offset by 100 points to the  
    /// left and 50 points below.  
    ///
```

```
    /// Circle()
    /// .overlay(
    ///     Circle()
    ///     .offset(CGPoint(x: -100,
y: 50))
    ///         .stroke()
    ///     )
    ///
/// - Parameter offset: The amount,
in points, by which you offset the
/// shape. Negative numbers are to
the left and up; positive numbers are
/// to the right and down.
///
/// - Returns: A shape offset by the
specified amount.
@inlinable public func offset(_  
offset: CGPoint) -> OffsetShape<Self>
```

```
    /// Changes the relative position of
this shape using the specified point.
///
/// The following example renders two
circles. It places one circle at its
/// default position. The second
circle is outlined with a stroke,
/// positioned on top of the first
circle and offset by 100 points to the
/// left and 50 points below.
///
/// Circle()
/// .overlay(
///     Circle()
```

```
    ///         .offset(x: -100, y: 50)
    ///         .stroke()
    ///     )
    ///
/// - Parameters:
///   - x: The horizontal amount, in
points, by which you offset the shape.
///   Negative numbers are to the
left and positive numbers are to the
///   right.
///   - y: The vertical amount, in
points, by which you offset the shape.
///   Negative numbers are up and
positive numbers are down.
///
/// - Returns: A shape offset by the
specified amount.
@inlinable public func offset(x:
CGFloat = 0, y: CGFloat = 0) ->
OffsetShape<Self>

    /// Scales this shape without
changing its bounding frame.
///
/// Both the `x` and `y`
multiplication factors halve their
respective
    /// dimension's size when set to
`0.5`, maintain their existing size when
    /// set to `1`, double their size
when set to `2`, and so forth.
///
/// - Parameters:
```

```
    /// - x: The multiplication factor  
used to resize this shape along its  
    ///     x-axis.  
    /// - y: The multiplication factor  
used to resize this shape along its  
    ///     y-axis.  
    ///  
    /// - Returns: A scaled form of this  
shape.  
    @inlinable public func scale(x:  
CGFloat = 1, y: CGFloat = 1, anchor:  
UnitPoint = .center) -> ScaledShape<Self>
```

```
    /// Scales this shape without  
changing its bounding frame.  
    ///  
    /// - Parameter scale: The  
multiplication factor used to resize this  
shape.  
    /// A value of `0` scales the shape  
to have no size, `0.5` scales to half  
    /// size in both dimensions, `2`  
scales to twice the regular size, and so  
    /// on.  
    ///  
    /// - Returns: A scaled form of this  
shape.  
    @inlinable public func scale(_ scale:  
CGFloat, anchor: UnitPoint = .center) ->  
ScaledShape<Self>
```

```
    /// Rotates this shape around an  
anchor point at the angle you specify.
```

```
///  
/// The following example rotates a  
square by 45 degrees to the right to  
/// create a diamond shape:  
///  
///  
///  
RoundedRectangle(cornerRadius: 10)  
    /// .rotation(Angle(degrees: 45))  
    /// .aspectRatio(1.0,  
contentMode: .fit)  
///  
/// - Parameters:  
/// - angle: The angle of rotation  
to apply. Positive angles rotate  
/// clockwise; negative angles  
rotate counterclockwise.  
/// - anchor: The point to rotate  
the shape around.  
///  
/// - Returns: A rotated shape.  
@inlinable public func rotation(_  
angle: Angle, anchor: UnitPoint  
= .center) -> RotatedShape<Self>  
  
/// Applies an affine transform to  
this shape.  
///  
/// Affine transforms present a  
mathematical approach to applying  
/// combinations of rotation,  
scaling, translation, and skew to shapes.  
///  
/// - Parameter transform: The affine
```

```
transformation matrix to apply to this
    ///      shape.
    ///
    /// - Returns: A transformed shape,
based on its matrix values.
    @inlinable public func transform(_
transform: CGAffineTransform) ->
TransformedShape<Self>
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Shape {

    /// Fills this shape with a color or
gradient.
    ///
    /// - Parameters:
    ///   - content: The color or
gradient to use when filling this shape.
    ///   - style: The style options that
determine how the fill renders.
    /// - Returns: A shape filled with
the color or gradient you supply.
    @inlinable nonisolated public func
fill<S>(_ content: S, style: FillStyle =
FillStyle()) -> some View where S :
ShapeStyle

    /// Fills this shape with the
foreground color.
    ///
```

```
    /// - Parameter style: The style  
options that determine how the fill  
    /// renders.
```

```
    /// - Returns: A shape filled with  
the foreground color.
```

```
@inlinable nonisolated public func  
fill(style: FillStyle = FillStyle()) ->  
some View
```

```
    /// Traces the outline of this shape  
with a color or gradient.
```

```
    ///  
    /// The following example adds a  
dashed purple stroke to a `Capsule`:
```

```
    ///  
    /// Capsule()  
    /// .stroke(  
    ///     Color.purple,  
    ///     style: StrokeStyle(  
    ///         lineWidth: 5,  
    ///         lineCap: .round,  
    ///         lineJoin: .miter,  
    ///         miterLimit: 0,  
    ///         dash: [5, 10],  
    ///         dashPhase: 0  
    ///     )  
    /// )  
    ///
```

```
    /// - Parameters:
```

```
    ///     - content: The color or  
gradient with which to stroke this shape.
```

```
    ///     - style: The stroke
```

characteristics --- such as the line's width and

    /// whether the stroke is dashed  
--- that determine how to render this

    /// shape.

    /// - Returns: A stroked shape.

```
@inlinable nonisolated public func
stroke<S>(_ content: S, style:
StrokeStyle) -> some View where S :
ShapeStyle
```

    /// Traces the outline of this shape with a color or gradient.

    ///

    /// The following example draws a circle with a purple stroke:

    ///

```
    /// Circle().stroke(Color.purple,
lineWidth: 5)
```

    ///

    /// - Parameters:

        /// - content: The color or gradient with which to stroke this shape.

        /// - lineWidth: The width of the stroke that outlines this shape.

    /// - Returns: A stroked shape.

```
@inlinable nonisolated public func
stroke<S>(_ content: S, lineWidth:
CGFloat = 1) -> some View where S :
ShapeStyle
```

}

```
/// A shape acts as view by filling
itself with the foreground color and
/// default fill style.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Shape {

    /// The content and behavior of the
view.
    ///
    /// When you implement a custom view,
you must implement a computed
    /// `body` property to provide the
content for your view. Return a view
    /// that's composed of built-in views
that SwiftUI provides, plus other
    /// composite views that you've
already defined:
    ///
    ///     struct MyView: View {
    ///         var body: some View {
    ///             Text("Hello, World!")
    ///         }
    ///     }
    ///
    /// For more information about
composing views and a view hierarchy,
    /// see <doc:Declaring-a-Custom-
View>.
    public var body: _ShapeView<Self,
ForegroundStyle> { get }
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Shape {  
  
    /// Fills this shape with a color or  
    gradient.  
    ///  
    /// - Parameters:  
    ///     - content: The color or  
    gradient to use when filling this shape.  
    ///     - style: The style options that  
    determine how the fill renders.  
    /// - Returns: A shape filled with  
    the color or gradient you supply.  
    nonisolated public func fill<S>(_  
content: S = .foreground, style:  
FillStyle = FillStyle()) ->  
_ShapeView<Self, S> where S : ShapeStyle  
  
    /// Traces the outline of this shape  
    with a color or gradient.  
    ///  
    /// The following example adds a  
    dashed purple stroke to a `Capsule`:  
    ///  
    ///     Capsule()  
    ///     .stroke(  
    ///         Color.purple,  
    ///         style: StrokeStyle(  
    ///             lineWidth: 5,  
    ///             lineCap: .round,  
    ///             lineJoin: .miter,
```

```
    /**
     * miterLimit: 0,
     * dash: [5, 10],
     * dashPhase: 0
     */
    )
}

/**
 * - Parameters:
 *   - content: The color or
gradient with which to stroke this shape.
 *   - style: The stroke
characteristics --- such as the line's
width and
 *   whether the stroke is dashed
--- that determine how to render this
 *   shape.
 * - Returns: A stroked shape.
nonisolated public func stroke<S>(_
content: S, style: StrokeStyle,
antialiased: Bool = true) ->
StrokeShapeView<Self, S, EmptyView> where
S : ShapeStyle

    /**
     * Traces the outline of this shape
with a color or gradient.
    */
    /**
     * The following example draws a
circle with a purple stroke:
    */
    /**
     * Circle().stroke(Color.purple,
lineWidth: 5)
    */
    /**
     * - Parameters:
     *   - content: The color or
```

```
gradient with which to stroke this shape.  
    /// - lineWidth: The width of the  
stroke that outlines this shape.  
    /// - Returns: A stroked shape.  
    nonisolated public func stroke<S>(_  
content: S, lineWidth: CGFloat = 1,  
antialiased: Bool = true) ->  
StrokeShapeView<Self, S, EmptyView> where  
S : ShapeStyle  
}  
  
/// Ways of styling a shape.  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
public enum ShapeRole : Sendable {  
  
    /// Indicates to the shape's style  
that SwiftUI fills the shape.  
    case fill  
  
    /// Indicates to the shape's style  
that SwiftUI applies a stroke to  
    /// the shape's path.  
    case stroke  
  
    /// Indicates to the shape's style  
that SwiftUI uses the shape as a  
    /// separator.  
    case separator  
  
    /// 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: ShapeRole,  
b: ShapeRole) -> Bool
```

```
    /// 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)

    /// 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 }
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension ShapeRole : Equatable {

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension ShapeRole : Hashable {
}
```

```
/// A color or pattern to use when
// rendering a shape.
///
/// You create custom shape styles by
// declaring a type that conforms to the
/// `ShapeStyle` protocol and
// implementing the required `resolve`
// function to
/// return a shape style that represents
// the desired appearance based on the
/// current environment.
///
/// For example this shape style reads
// the current color scheme from the
/// environment to choose the blend mode
// its color will be composited with:
///
///     struct MyShapeStyle: ShapeStyle {
///         func resolve(in environment:
EnvironmentValues) -> some ShapeStyle {
///             if
environment.colorScheme == .light {
///                 return
Color.red.blendMode(.lighten)
///             } else {
///                 return
Color.red.blendMode(.darken)
///             }
///         }
///     }
///
/// In addition to creating a custom
```

shape style, you can also use one of the  
/// concrete styles that SwiftUI defines.  
To indicate a specific color or  
/// pattern, you can use ``Color`` or the  
style returned by  
///  
``ShapeStyle/image(\_:sourceRect:scale:)``  
, or one of the gradient  
/// types, like the one returned by  
///  
``ShapeStyle/radialGradient(\_:center:star  
tRadius:endRadius:)``.  
/// To set a color that's appropriate for  
a given context on a given  
/// platform, use one of the semantic  
styles, like ``ShapeStyle/background`` or  
/// ``ShapeStyle/primary``.  
///  
/// You can use a shape style by:  
/// \* Filling a shape with a style with  
the ``Shape/fill(\_:style:)``  
/// modifier:  
///  
///   ``  
///   Path { path in  
///       path.move(to: .zero)  
///       path.addLine(to: CGPoint(x:  
50, y: 0))  
///       path.addArc(  
///           center: .zero,  
///           radius: 50,  
///           startAngle: .zero,  
///           endAngle: .degrees(90),

```
///           clockwise: false)
///     }
///     .fill(.radialGradient(
///         Gradient(colors:
/// [.yellow, .red]),
///         center: .topLeading,
///         startRadius: 15,
///         endRadius: 80))
///     ...
///
///     ! [A screenshot of a quarter of a
circle filled with
///     a radial gradient.] (ShapeStyle-1)
///
/// * Tracing the outline of a shape with
a style with either the
///     ``Shape/stroke(_:lineWidth:)`` or
the ``Shape/stroke(_:style:)`` modifier:
///
///     ...
///
///     RoundedRectangle(cornerRadius:
10)
///             .stroke(.mint, lineWidth: 10)
///             .frame(width: 200, height:
50)
///             ...
///
///     ! [A screenshot of a rounded
rectangle, outlined in mint.]
(ShapeStyle-2)
///
/// * Styling the foreground elements in
a view with the
```

```
/// ``View/foregroundStyle(_:)``
modifier:
///
///
VStack(alignment: .leading) {
    Text("Primary")
        .font(.title)
    Text("Secondary")
        .font(.caption)
        .foregroundStyle(.seconda
ry)
}
```
```
![A screenshot of a title in the primary content color above a subtitle in the secondary content color.](ShapeStyle-3)

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public protocol ShapeStyle : Sendable {

    /// The type of shape style this will
    resolve to.
    ///
    /// When you create a custom shape
    style, Swift infers this type
    /// from your implementation of the
    required `resolve` function.
    @available(iOS 17.0, macOS 14.0, tvOS
    17.0, watchOS 10.0, *)
    associatedtype Resolved : ShapeStyle
= Never
```

```
    /// Evaluate to a resolved shape
    style given the current `environment`.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    func resolve(in environment:
EnvironmentValues) -> Self.Resolved
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ShapeStyle where Self ==
LinearGradient {
    /// A linear gradient.
    ///
    /// The gradient applies the color
    function along an axis, as defined by its
    /// start and end points. The
    gradient maps the unit space points into
    the
    /// bounding rectangle of each shape
    filled with the gradient.
    ///
    /// For information about how to use
    shape styles, see ``ShapeStyle``.
    public static func linearGradient(_
gradient: Gradient, startPoint:
UnitPoint, endPoint: UnitPoint) ->
LinearGradient
    /// A linear gradient defined by a
    collection of colors.
```

```
///  
/// The gradient applies the color  
function along an axis, as defined by its  
/// start and end points. The  
gradient maps the unit space points into  
the  
    /// bounding rectangle of each shape  
filled with the gradient.  
    ///  
    /// For information about how to use  
shape styles, see ``ShapeStyle``.  
    public static func  
linearGradient(colors: [Color],  
startPoint: UnitPoint, endPoint:  
UnitPoint) -> LinearGradient  
  
    /// A linear gradient defined by a  
collection of color stops.  
    ///  
    /// The gradient applies the color  
function along an axis, as defined by its  
/// start and end points. The  
gradient maps the unit space points into  
the  
    /// bounding rectangle of each shape  
filled with the gradient.  
    ///  
    /// For information about how to use  
shape styles, see ``ShapeStyle``.  
    public static func  
linearGradient(stops: [Gradient.Stop],  
startPoint: UnitPoint, endPoint:  
UnitPoint) -> LinearGradient
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ShapeStyle where Self ==  
RadialGradient {  
  
    /// A radial gradient.  
    ///  
    /// The gradient applies the color  
    /// function as the distance from a center  
    /// point, scaled to fit within the  
    /// defined start and end radii. The  
    /// gradient maps the unit space  
    /// center point into the bounding rectangle  
    /// of  
    /// each shape filled with the  
    /// gradient.  
    ///  
    /// For information about how to use  
    /// shape styles, see ``ShapeStyle``.  
    public static func radialGradient(_  
gradient: Gradient, center: UnitPoint,  
startRadius: CGFloat, endRadius: CGFloat)  
-> RadialGradient  
  
    /// A radial gradient defined by a  
    /// collection of colors.  
    ///  
    /// The gradient applies the color  
    /// function as the distance from a center  
    /// point, scaled to fit within the  
    /// defined start and end radii. The
```

```
    /// gradient maps the unit space
    center point into the bounding rectangle
    of
        /// each shape filled with the
        gradient.
        ///
        /// For information about how to use
        shape styles, see ``ShapeStyle``.
    public static func
radialGradient(colors: [Color], center:
UnitPoint, startRadius: CGFloat,
endRadius: CGFloat) -> RadialGradient

    /// A radial gradient defined by a
    collection of color stops.
    ///
    /// The gradient applies the color
    function as the distance from a center
    /// point, scaled to fit within the
    defined start and end radii. The
    /// gradient maps the unit space
    center point into the bounding rectangle
    of
        /// each shape filled with the
        gradient.
        ///
        /// For information about how to use
        shape styles, see ``ShapeStyle``.
    public static func
radialGradient(stops: [Gradient.Stop],
center: UnitPoint, startRadius: CGFloat,
endRadius: CGFloat) -> RadialGradient
}
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension ShapeStyle where Self ==  
EllipticalGradient {  
  
    /// A radial gradient that draws an  
    /// ellipse.  
    ///  
    /// The gradient maps its coordinate  
    /// space to the unit space square  
    /// in which its center and radii are  
    /// defined, then stretches that  
    /// square to fill its bounding rect,  
    /// possibly also stretching the  
    /// circular gradient to have  
    /// elliptical contours.  
    ///  
    /// For example, an elliptical  
    /// gradient used as a background:  
    ///  
    ///     let gradient =  
    Gradient(colors: [.red, .yellow])  
    ///  
    ///     ContentView()  
    ///         .background(.ellipticalGr  
adient(gradient))  
    ///  
    /// For information about how to use  
    /// shape styles, see ``ShapeStyle``.  
    public static func  
ellipticalGradient(_ gradient: Gradient,  
center: UnitPoint = .center,
```

```
startRadiusFraction: CGFloat = 0,  
endRadiusFraction: CGFloat = 0.5) ->  
EllipticalGradient
```

```
    /// A radial gradient that draws an  
    ellipse defined by a collection of  
    /// colors.  
    ///  
    /// The gradient maps its coordinate  
    space to the unit space square  
    /// in which its center and radii are  
    defined, then stretches that  
    /// square to fill its bounding rect,  
    possibly also stretching the  
    /// circular gradient to have  
    elliptical contours.  
    ///  
    /// For example, an elliptical  
    gradient used as a background:  
    ///  
    ///     .background(.elliptical(colors: [.red, .yellow]))  
    ///  
    /// For information about how to use  
    shape styles, see ``ShapeStyle``.  
    public static func  
ellipticalGradient(colors: [Color],  
center: UnitPoint = .center,  
startRadiusFraction: CGFloat = 0,  
endRadiusFraction: CGFloat = 0.5) ->  
EllipticalGradient
```

```
    /// A radial gradient that draws an
```

```
ellipse defined by a collection of
    /// color stops.
    ///
    /// The gradient maps its coordinate
    space to the unit space square
    /// in which its center and radii are
    defined, then stretches that
    /// square to fill its bounding rect,
    possibly also stretching the
    /// circular gradient to have
    elliptical contours.
    ///
    /// For example, an elliptical
    gradient used as a background:
    ///
    ///     .background(.ellipticalGradie
    nt(stops: [
        ///         .init(color: .red,
    location: 0.0),
        ///         .init(color: .yellow,
    location: 0.9),
        ///         .init(color: .yellow,
    location: 1.0),
        ///     ]))
    ///
    /// For information about how to use
    shape styles, see ``ShapeStyle``.

    public static func
ellipticalGradient(stops:
    [Gradient.Stop], center: UnitPoint
= .center, startRadiusFraction: CGFloat =
0, endRadiusFraction: CGFloat = 0.5) ->
EllipticalGradient
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ShapeStyle where Self ==  
AngularGradient {  
  
    /// An angular gradient, which  
    // applies the color function as the angle  
    /// changes between the start and end  
    angles, and anchored to a relative  
    /// center point within the filled  
    shape.  
    ///  
    /// An angular gradient is also known  
    as a "conic" gradient. If  
    /// `endAngle - startAngle > 2π`, the  
    gradient only draws the last complete  
    /// turn. If `endAngle - startAngle <  
    2π`, the gradient fills the missing  
    /// area with the colors defined by  
    gradient stop locations at `0` and `1`,  
    /// transitioning between the two  
    halfway across the missing area.  
    ///  
    /// For example, an angular gradient  
    used as a background:  
    ///  
    ///     let gradient =  
    Gradient(colors: [.red, .yellow])  
    ///  
    ///     ContentView()  
    ///         .background(.angularGradi
```

```
ent(gradient))
    /**
     /// For information about how to use
shape styles, see ``ShapeStyle``.
    /**
     /// - Parameters:
     ///   - gradient: The gradient to use
for filling the shape, providing the
     ///   colors and their relative
stop locations.
     ///   - center: The relative center
of the gradient, mapped from the unit
     ///   space into the bounding
rectangle of the filled shape.
     ///   - startAngle: The angle that
marks the beginning of the gradient.
     ///   - endAngle: The angle that
marks the end of the gradient.
public static func angularGradient(_
gradient: Gradient, center: UnitPoint,
startAngle: Angle, endAngle: Angle) ->
AngularGradient

    /// An angular gradient defined by a
collection of colors.
    /**
     /// For more information on how to
use angular gradients, see
    /**
````ShapeStyle/angularGradient(_:center:sta
rtAngle:endAngle:)````.
    /**
     /// - Parameters:
```

```
    /// - colors: The colors of the
    gradient, evenly spaced along its full
    /// length.
    /// - center: The relative center
    of the gradient, mapped from the unit
    /// space into the bounding
    rectangle of the filled shape.
    /// - startAngle: The angle that
    marks the beginning of the gradient.
    /// - endAngle: The angle that
    marks the end of the gradient.
public static func
angularGradient(colors: [Color], center:
UnitPoint, startAngle: Angle, endAngle:
Angle) -> AngularGradient
```

```
    /// An angular gradient defined by a
    collection of color stops.
    ///
    /// For more information on how to
    use angular gradients, see
    ///
``ShapeStyle/angularGradient(_:center:sta
rtAngle:endAngle:)``.
    ///
    /// - Parameters:
    /// - stops: The color stops of the
    gradient, defining each component
    /// color and their relative
    location along the gradient's full
    length.
    /// - center: The relative center
    of the gradient, mapped from the unit
```

```
    ///      space into the bounding
    rectangle of the filled shape.
    /// - startAngle: The angle that
    marks the beginning of the gradient.
    /// - endAngle: The angle that
    marks the end of the gradient.
    public static func
angularGradient(stops: [Gradient.Stop],
center: UnitPoint, startAngle: Angle,
endAngle: Angle) -> AngularGradient
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ShapeStyle where Self ==
AngularGradient {

    /// A conic gradient that completes a
    full turn, optionally starting from
    /// a given angle and anchored to a
    relative center point within the filled
    /// shape.
    ///
    /// For example, a conic gradient
    used as a background:
    ///
    ///     let gradient =
    Gradient(colors: [.red, .yellow])
    ///
    ///     ContentView()
    ///         .background(.conicGradien
t(gradient))
    ///
```

```
    /// For information about how to use
    shape styles, see ``ShapeStyle``.
    ///
    /// - Parameters:
    ///   - gradient: The gradient to use
    for filling the shape, providing the
    ///   colors and their relative
    stop locations.
    ///   - center: The relative center
    of the gradient, mapped from the unit
    ///   space into the bounding
    rectangle of the filled shape.
    ///   - angle: The angle to offset
    the beginning of the gradient's full
    ///   turn.
    public static func conicGradient(_
gradient: Gradient, center: UnitPoint,
angle: Angle = .zero) -> AngularGradient

    /// A conic gradient defined by a
    collection of colors that completes a
    full
    /// turn.
    ///
    /// For more information on how to
    use conic gradients, see
    ///
    ``ShapeStyle/conicGradient(_:center:angle
:)``.
    ///
    /// - Parameters:
    ///   - colors: The colors of the
    gradient, evenly spaced along its full
```

```
    /// length.  
    /// - center: The relative center  
of the gradient, mapped from the unit  
    /// space into the bounding  
rectangle of the filled shape.  
    /// - angle: The angle to offset  
the beginning of the gradient's full  
    /// turn.  
    public static func  
conicGradient(colors: [Color], center:  
UnitPoint, angle: Angle = .zero) ->  
AngularGradient  
  
    /// A conic gradient defined by a  
collection of color stops that completes  
a  
    /// full turn.  
    ///  
    /// For more information on how to  
use conic gradients, see  
    ///  
``ShapeStyle/conicGradient(_:center:angle  
:)``.  
    ///  
    /// - Parameters:  
    /// - stops: The color stops of the  
gradient, defining each component  
    /// color and their relative  
location along the gradient's full  
length.  
    /// - center: The relative center  
of the gradient, mapped from the unit  
    /// space into the bounding
```

```
rectangle of the filled shape.

    /// - angle: The angle to offset
the beginning of the gradient's full
    /// turn.

public static func
conicGradient(stops: [Gradient.Stop],
center: UnitPoint, angle: Angle = .zero)
-> AngularGradient
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension ShapeStyle where Self ==
LinearGradient {

    /// A linear gradient.

    ///
    /// The gradient applies the color
function along an axis, as
    /// defined by its start and end
points. The gradient maps the unit
    /// space points into the bounding
rectangle of each shape filled
    /// with the gradient.

    ///
    /// For example, a linear gradient
used as a background:
    ///
    ///     ContentView()
    ///         .background(.linearGradie
nt(.red.gradient,
    ///             startPoint: .top,
endPoint: .bottom))
```

```
    /**
     * For information about how to use
     * shape styles, see ``ShapeStyle``.
     */
    public static func linearGradient(_
gradient: AnyGradient, startPoint:
UnitPoint, endPoint: UnitPoint) -> some
ShapeStyle

}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension ShapeStyle where Self ==
RadialGradient {

    /**
     * A radial gradient.
     */
    /**
     * The gradient applies the color
     * function as the distance from a
     * center point, scaled to fit
     * within the defined start and end
     * radii. The gradient maps the unit
     * space center point into the
     * bounding rectangle of each shape
     * filled with the gradient.
     */
    /**
     * For example, a radial gradient
     * used as a background:
     */
    /**
     *      ContentView()
     *          .background(.radialGradie
     nt(.red.gradient, endRadius: 100))
     */
```

```
    /// For information about how to use
    shape styles, see ``ShapeStyle``.
    public static func radialGradient(_
gradient: AnyGradient, center: UnitPoint
= .center, startRadius: CGFloat = 0,
endRadius: CGFloat) -> some ShapeStyle

}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension ShapeStyle where Self ==
EllipticalGradient {

    /// A radial gradient that draws an
    ellipse.
    ///
    /// The gradient maps its coordinate
    space to the unit space square
    /// in which its center and radii are
    defined, then stretches that
    /// square to fill its bounding rect,
    possibly also stretching the
    /// circular gradient to have
    elliptical contours.
    ///
    /// For example, an elliptical
    gradient used as a background:
    ///
    ///     ContentView()
    ///         .background(.ellipticalGr
adient(.red.gradient))
    ///
```

```
    /// For information about how to use
    shape styles, see ``ShapeStyle``.
    public static func
    ellipticalGradient(_ gradient:
    AnyGradient, center: UnitPoint = .center,
    startRadiusFraction: CGFloat = 0,
    endRadiusFraction: CGFloat = 0.5) -> some
    ShapeStyle

}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension ShapeStyle where Self ==
AngularGradient {

    /// An angular gradient, which
    applies the color function as the
    /// angle changes between the start
    and end angles, and anchored to
    /// a relative center point within
    the filled shape.
    ///
    /// An angular gradient is also known
    as a "conic" gradient. If
    /// `endAngle - startAngle > 2π`, the
    gradient only draws the last complete
    /// turn. If `endAngle - startAngle <
    2π`, the gradient fills the missing
    /// area with the colors defined by
    gradient stop locations at `0` and `1`,
    /// transitioning between the two
    halfway across the missing area.
```

```
///  
/// For example, an angular gradient  
used as a background:  
///  
///     ContentView()  
///         .background(.angularGradi  
ent(.red.gradient))  
///  
/// For information about how to use  
shape styles, see ``ShapeStyle``.  
///  
/// - Parameters:  
///   - gradient: The gradient to use  
for filling the shape, providing the  
///   colors and their relative  
stop locations.  
///   - center: The relative center  
of the gradient, mapped from the unit  
///   space into the bounding  
rectangle of the filled shape.  
///   - startAngle: The angle that  
marks the beginning of the gradient.  
///   - endAngle: The angle that  
marks the end of the gradient.  
public static func angularGradient(_  
gradient: AnyGradient, center: UnitPoint  
= .center, startAngle: Angle, endAngle:  
Angle) -> some ShapeStyle
```

```
/// A conic gradient that completes a  
full turn, optionally starting from  
/// a given angle and anchored to a
```

```
relative center point within the filled
    /// shape.
    ///
    /// For example, a conic gradient
used as a background:
    ///
    ///     let gradient =
Gradient(colors: [.red, .yellow])
    ///
    ///     ContentView()
    ///         .background(.conicGradien
t(gradient))
    ///
    /// For information about how to use
shape styles, see ``ShapeStyle``.
    ///
    /// - Parameters:
    ///   - gradient: The gradient to use
for filling the shape, providing the
    ///   colors and their relative
stop locations.
    ///   - center: The relative center
of the gradient, mapped from the unit
    ///   space into the bounding
rectangle of the filled shape.
    ///   - angle: The angle to offset
the beginning of the gradient's full
    ///   turn.
public static func conicGradient(_
gradient: AnyGradient, center: UnitPoint
= .center, angle: Angle = .zero) -> some
ShapeStyle
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ShapeStyle where Self ==  
ImagePaint {  
  
    /// A shape style that fills a shape  
    // by repeating a region of an image.  
    ///  
    /// For information about how to use  
    // shape styles, see ``ShapeStyle``.  
    ///  
    /// - Parameters:  
    ///     - image: The image to be drawn.  
    ///     - sourceRect: A unit-space  
    // rectangle defining how much of the source  
    ///     image to draw. The results  
    // are undefined if `sourceRect` selects  
    ///     areas outside the `[0, 1]`  
    range in either axis.  
    ///     - scale: A scale factor applied  
    to the image during rendering.  
    public static func image(_ image:  
Image, sourceRect: CGRect = CGRect(x: 0,  
y: 0, width: 1, height: 1), scale:  
CGFloat = 1) -> ImagePaint  
}  
  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension ShapeStyle where Self ==  
AnyShapeStyle {
```

```
    /// Returns a new style based on the
    current style that multiplies
    /// by `opacity` when drawing.
    ///
    /// In most contexts the current
    style is the foreground but e.g.
    /// when setting the value of the
    background style, that becomes
    /// the current implicit style.
    ///
    /// For example, a circle filled with
    the current foreground
    /// style at fifty-percent opacity:
    ///
    ///     Circle().fill(.opacity(0.5))
    ///
    public static func opacity(_ opacity:
Double) -> some ShapeStyle
```

```
}
```

```
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension ShapeStyle {
```

```
    /// Returns a new style based on
    `self` that applies the specified
    /// blend mode when drawing.
    @inlinable public func blendMode(_
mode: BlendMode) -> some ShapeStyle
```

```
}
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension ShapeStyle where Self ==  
AnyShapeStyle {  
  
    /// Returns a new style based on the  
    /// current style that uses  
    /// `mode` as its blend mode when  
    /// drawing.  
    ///  
    /// In most contexts the current  
    /// style is the foreground but e.g.  
    /// when setting the value of the  
    /// background style, that becomes  
    /// the current implicit style.  
    ///  
    /// For example, a circle filled with  
    /// the current foreground  
    /// style and the overlay blend mode:  
    ///  
    ///  
    Circle().fill(.blendMode(.overlay))  
    ///  
    public static func blendMode(_ mode:  
BlendMode) -> some ShapeStyle  
  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 10.0, *)  
extension ShapeStyle where Self ==  
Material {
```

```
    /// A material that's somewhat
    translucent.
    public static var regularMaterial:
Material { get }

    /// A material that's more opaque
    than translucent.
    public static var thickMaterial:
Material { get }

    /// A material that's more
    translucent than opaque.
    public static var thinMaterial:
Material { get }

    /// A mostly translucent material.
    public static var ultraThinMaterial:
Material { get }

    /// A mostly opaque material.
    public static var ultraThickMaterial:
Material { get }
}

@available(iOS 15.0, macOS 12.0, *)
@available(tvOS, unavailable)
@available(watchOS, unavailable)
extension ShapeStyle where Self ==
Material {

    /// A material matching the style of
    system toolbars.
```

```
    public static var bar: Material { get
}

}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension ShapeStyle {

    /// Sets an explicit active
    appearance for materials created by this
    style.
    public func
    materialActiveAppearance(_ appearance:
    MaterialActiveAppearance) -> some
    ShapeStyle
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension ShapeStyle where Self ==
HierarchicalShapeStyle {

    /// A shape style that maps to the
    first level of the current content style.
    ///
    /// This hierarchical style maps to
    the first level of the current
    /// foreground style, or to the first
    level of the default foreground style
    /// if you haven't set a foreground
    style in the view's environment. You
    /// typically set a foreground style
```

```
by supplying a non-hierarchical style
    /// to the
``View/foregroundStyle(_:)`` modifier.
    ///
    /// For information about how to use
shape styles, see ``ShapeStyle``.
public static var primary:
HierarchicalShapeStyle { get }

    /// A shape style that maps to the
second level of the current content
style.
    ///
    /// This hierarchical style maps to
the second level of the current
    /// foreground style, or to the
second level of the default foreground
style
    /// if you haven't set a foreground
style in the view's environment. You
    /// typically set a foreground style
by supplying a non-hierarchical style
    /// to the
``View/foregroundStyle(_:)`` modifier.
    ///
    /// For information about how to use
shape styles, see ``ShapeStyle``.
public static var secondary:
HierarchicalShapeStyle { get }

    /// A shape style that maps to the
third level of the current content
    /// style.
```

```
///  
/// This hierarchical style maps to  
the third level of the current  
/// foreground style, or to the third  
level of the default foreground style  
/// if you haven't set a foreground  
style in the view's environment. You  
/// typically set a foreground style  
by supplying a non-hierarchical style  
/// to the  
``View/foregroundStyle(_:)`` modifier.  
///  
/// For information about how to use  
shape styles, see ``ShapeStyle``.  
public static var tertiary:  
HierarchicalShapeStyle { get }  
  
/// A shape style that maps to the  
fourth level of the current content  
/// style.  
///  
/// This hierarchical style maps to  
the fourth level of the current  
/// foreground style, or to the  
fourth level of the default foreground  
style  
/// if you haven't set a foreground  
style in the view's environment. You  
/// typically set a foreground style  
by supplying a non-hierarchical style  
/// to the  
``View/foregroundStyle(_:)`` modifier.  
///
```

```
    /// For information about how to use
    shape styles, see ``ShapeStyle``.
    public static var quaternary:
HierarchicalShapeStyle { get }
}

@available(iOS 16.0, macOS 12.0,
macCatalyst 15.0, tvOS 17.0, watchOS
10.0, *)
extension ShapeStyle where Self ==
HierarchicalShapeStyle {

    /// A shape style that maps to the
    fifth level of the current content
    /// style.
    ///
    /// This hierarchical style maps to
    the fifth level of the current
    /// foreground style, or to the fifth
    level of the default foreground style
    /// if you haven't set a foreground
    style in the view's environment. You
    /// typically set a foreground style
    by supplying a non-hierarchical style
    /// to the
    ``View/foregroundStyle(_:)`` modifier.
    ///
    /// For information about how to use
    shape styles, see ``ShapeStyle``.
    public static var quinary:
HierarchicalShapeStyle { get }
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension ShapeStyle {  
  
    /// Returns the second level of this  
    shape style.  
    public var secondary: some ShapeStyle  
    { get }  
  
    /// Returns the third level of this  
    shape style.  
    public var tertiary: some ShapeStyle  
    { get }  
  
    /// Returns the fourth level of this  
    shape style.  
    public var quaternary: some  
    ShapeStyle { get }  
  
    /// Returns the fifth level of this  
    shape style.  
    public var quinary: some ShapeStyle {  
get }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ShapeStyle where Self == Color  
{  
  
    /// A context-dependent red color  
    suitable for use in UI elements.  
    public static var red: Color { get }
```

```
    /// A context-dependent orange color  
    suitable for use in UI elements.  
    public static var orange: Color { get }  
  
    /// A context-dependent yellow color  
    suitable for use in UI elements.  
    public static var yellow: Color { get }  
  
    /// A context-dependent green color  
    suitable for use in UI elements.  
    public static var green: Color {  
        get }  
  
    /// A context-dependent mint color  
    suitable for use in UI elements.  
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    public static var mint: Color { get }  
  
    /// A context-dependent teal color  
    suitable for use in UI elements.  
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    public static var teal: Color { get }  
  
    /// A context-dependent cyan color  
    suitable for use in UI elements.  
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    public static var cyan: Color { get }
```

```
    /// A context-dependent blue color  
    suitable for use in UI elements.  
    public static var blue: Color { get }  
  
    /// A context-dependent indigo color  
    suitable for use in UI elements.  
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    public static var indigo: Color { get }  
  
    /// A context-dependent purple color  
    suitable for use in UI elements.  
    public static var purple: Color { get }  
  
    /// A context-dependent pink color  
    suitable for use in UI elements.  
    public static var pink: Color { get }  
  
    /// A context-dependent brown color  
    suitable for use in UI elements.  
    @available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
    public static var brown: Color {  
get }  
  
    /// A white color suitable for use in  
    UI elements.  
    public static var white: Color {  
get }
```

```
    /// A context-dependent gray color  
    suitable for use in UI elements.  
    public static var gray: Color { get }  
  
    /// A black color suitable for use in  
    UI elements.  
    public static var black: Color {  
        get }  
  
    /// A clear color suitable for use in  
    UI elements.  
    public static var clear: Color {  
        get }  
    }  
  
@available(iOS 17.0, macOS 10.15, tvOS  
17.0, watchOS 10.0, *)  
extension ShapeStyle where Self ==  
SeparatorShapeStyle {  
  
    /// A style appropriate for  
    foreground separator or border lines.  
    ///  
    /// For information about how to use  
    shape styles, see ``ShapeStyle``.  
    public static var separator:  
SeparatorShapeStyle { get }  
}  
  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension ShapeStyle {
```

```
    /// Applies the specified shadow
    effect to the shape style.
    /**
     * For example, you can create a
     * rectangle that adds a drop shadow to
     * the ``ShapeStyle/red`` shape
     * style.
    /**
    /**
    Rectangle().fill(.red.shadow(.drop(radius
        : 2, y: 3)))
    /**
     * - Parameter style: The shadow
     * style to apply.
    /**
     * - Returns: A new shape style that
     * uses the specified shadow style.
    @inlinable public func shadow(_
style: ShadowStyle) -> some ShapeStyle
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension ShapeStyle where Self == AnyShapeStyle {

    /**
     * Returns a shape style that
     * applies the specified shadow style to the
     * current style.
    /**
     * In most contexts the current
     * style is the foreground, but not always.
}
```

```
    /// For example, when setting the
    value of the background style, that
    /// becomes the current implicit
    style.
    ///
    /// The following example creates a
    circle filled with the current
    /// foreground style that uses an
    inner shadow:
    ///
    ///
Circle().fill(.shadow(.inner(radius: 1,
y: 1)))
    ///
    /// - Parameter style: The shadow
    style to apply.
    ///
    /// - Returns: A new shape style
    based on the current style that uses the
    /// specified shadow style.
    public static func shadow(_ style:
ShadowStyle) -> some ShapeStyle

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ShapeStyle {

    /// Maps a shape style's unit-space
    coordinates to the absolute coordinates
    /// of a given rectangle.
    ///
```

```
    /// Some shape styles have colors or
patterns that vary
    /// with position based on
``UnitPoint`` coordinates. For example,
you
    /// can create a ``LinearGradient``
using ``UnitPoint/top`` and
    /// ``UnitPoint/bottom`` as the start
and end points:
    ///
    ///     let gradient =
LinearGradient(
    ///         colors: [.red, .yellow],
    ///         startPoint: .top,
    ///         endPoint: .bottom)
    ///
    /// When rendering such styles,
SwiftUI maps the unit space coordinates
to
    /// the absolute coordinates of the
filled shape. However, you can tell
    /// SwiftUI to use a different set of
coordinates by supplying a rectangle
    /// to the `in(_:)` method. Consider
two resizable rectangles using the
    /// gradient defined above:
    ///
    ///     HStack {
    ///         Rectangle()
    ///             .fill(gradient)
    ///         Rectangle()
    ///             .fill(gradient.in(CGRect(
ect(x: 0, y: 0, width: 0, height: 300))))
```

```
    /**
     */
    .onTapGesture
{ isBig.toggle() }
    /**
     .frame(height: isBig ? 300 :
50)
    /**
     .animation(.easeInOut)
    /**
     /**
      When `isBig` is true – defined
      elsewhere as a private ``State``
      // variable – the rectangles look
      the same, because their heights
      // match that of the modified
      gradient:
    /**
      /**
       ! [Two identical, tall rectangles,
       with a gradient that starts red at
       // the top and transitions to yellow
       at the bottom.] (ShapeStyle-in-1)
    /**
      /**
       When the user toggles `isBig` by
       tapping the ``HStack``, the
       // rectangles shrink, but the
       gradients each react in a different way:
    /**
      /**
       ! [Two short rectangles with
       different coloration. The first has a
       // gradient that transitions top to
       bottom from full red to full yellow.
       // The second starts as red at the
       top and then begins to transition
       // to yellow toward the bottom.] (ShapeStyle-in-2)
    /**

```

```
    /// SwiftUI remaps the gradient of
    the first rectangle to the new frame
    /// height, so that you continue to
    see the full range of colors in a
    /// smaller area. For the second
    rectangle, the modified gradient retains
    /// a mapping to the full height, so
    you instead see only a small part of
    /// the overall gradient. Animation
    helps to visualize the difference.

    ///
    /// - Parameter rect: A rectangle
    that gives the absolute coordinates over
    /// which to map the shape style.
    /// - Returns: A new shape style
    mapped to the coordinates given by
    `rect`.

    @inlinable public func `in`(_ rect:
    CGRect) -> some ShapeStyle
```

}

```
@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension ShapeStyle where Self ==
BackgroundStyle {
```

```
    /// The background style in the
    current context.
    ///
    /// Access this value to get the
    style SwiftUI uses for the background
    /// in the current context. The
```

specific color that SwiftUI renders depends

```
    /// on factors like the platform and
    /// whether the user has turned on Dark
    /// Mode.
    ///
    /// For information about how to use
    /// shape styles, see ``ShapeStyle``.
    public static var background:
BackgroundStyle { get }
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ShapeStyle where Self ==
ForegroundStyle {
```

/// The foreground style in the current context.

```
    ///
    /// Access this value to get the style SwiftUI uses for foreground elements,
```

/// like text, symbols, and shapes, in the current context. Use the

```
    /// ``View/foregroundStyle(_:)``
    modifier to set a new foreground style for
```

/// a given view and its child views.

```
    ///
    /// For information about how to use
```

/// shape styles, see ``ShapeStyle``.

```
    public static var foreground:
```

```
ForegroundStyle { get }  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension ShapeStyle where Self.Resolved  
== Never {  
  
    /// Evaluate to a resolved shape  
    style given the current `environment`.  
    public func resolve(in environment:  
EnvironmentValues) -> Never  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ShapeStyle where Self : View,  
Self.Body == _ShapeView<Rectangle, Self>  
{  
  
    /// A rectangular view that's filled  
    with the shape style.  
    ///  
    /// For a ``ShapeStyle`` that also  
    conforms to the ``View`` protocol, like  
    /// ``Color`` or ``LinearGradient``,  
    this default implementation of the  
    /// ``View/body-swift.property``  
    property provides a visual representation  
    /// for the shape style. As a result,  
    you can use the shape style in a view  
    /// hierarchy like any other view:  
    ///
```

```
    ///     ZStack {
    ///         Color.cyan
    ///         Text("Hello!")
    ///     }
    ///     .frame(width: 200, height:
50)
    ///
    /// ! [A screenshot of a cyan
rectangle with the text hello appearing
    /// in the middle of the rectangle.]
```

(ShapeStyle-body-1)

```
    public var body:
ShapeView<Rectangle, Self> { get }
```

```
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension ShapeStyle where Self ==
TintShapeStyle {
```

```
    /// A style that reflects the current
tint color.
```

```
    ///
    /// You can set the tint color with
the `tint(_:)` modifier. If no explicit
    /// tint is set, the tint is derived
from the app's accent color.
```

```
    public static var tint:
TintShapeStyle { get }
```

```
}
```

```
/// A view that provides a shape that you
can use for drawing operations.
```

```
///  
/// Use this type with the drawing  
methods on ``Shape`` to apply multiple  
fills  
/// and/or strokes to a shape. For  
example, the following code applies a  
fill  
/// and stroke to a capsule shape:  
///  
///     Capsule()  
///         .fill(.yellow)  
///         .stroke(.blue, lineWidth: 8)  
///  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
public protocol ShapeView<Content> : View  
{  
  
    /// The type of shape this can  
provide.  
    associatedtype Content : Shape  
  
    /// The shape that this type draws  
and provides for other drawing  
    /// operations.  
    var shape: Self.Content { get }  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension ShapeView {  
  
    /// Fills this shape with a color or
```

gradient.

///

/// - Parameters:

/// - content: The color or gradient to use when filling this shape.

/// - style: The style options that determine how the fill renders.

/// - Returns: A shape filled with the color or gradient you supply.

```
nonisolated public func fill<S>(_  
content: S = .foreground, style:  
FillStyle = FillStyle()) ->  
FillShapeView<Self.Content, S, Self>  
where S : ShapeStyle
```

/// Traces the outline of this shape with a color or gradient.

///

/// The following example adds a dashed purple stroke to a `Capsule`:

///

///

Capsule()

///

.stroke(

///

Color.purple,

///

style: StrokeStyle(

///

lineWidth: 5,

///

lineCap: .round,

///

lineJoin: .miter,

///

miterLimit: 0,

///

dash: [5, 10],

///

dashPhase: 0

/// )

/// )

```
///  
/// - Parameters:  
///   - content: The color or  
gradient with which to stroke this shape.  
///   - style: The stroke  
characteristics --- such as the line's  
width and  
///     whether the stroke is dashed  
--- that determine how to render this  
///     shape.  
/// - Returns: A stroked shape.  
nonisolated public func stroke<S>(_  
content: S, style: StrokeStyle,  
antialiased: Bool = true) ->  
StrokeShapeView<Self.Content, S, Self>  
where S : ShapeStyle  
  
    /// Traces the outline of this shape  
with a color or gradient.  
///  
    /// The following example draws a  
circle with a purple stroke:  
///  
    ///     Circle().stroke(Color.purple,  
lineWidth: 5)  
///  
    /// - Parameters:  
    ///   - content: The color or  
gradient with which to stroke this shape.  
    ///   - lineWidth: The width of the  
stroke that outlines this shape.  
    /// - Returns: A stroked shape.  
nonisolated public func stroke<S>(_
```

```
content: S, lineWidth: CGFloat = 1,  
antialiased: Bool = true) ->  
StrokeShapeView<Self.Content, S, Self>  
where S : ShapeStyle  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension ShapeView where Self.Content :  
InsettableShape {  
  
    /// Returns a view that's the result  
    of insetting this view by half of its  
    style's line width.  
    ///  
    /// This method strokes the resulting  
    shape with  
    /// `style` and fills it with  
    `content`.  
    nonisolated public func  
strokeBorder<S>(_ content: S  
= .foreground, style: StrokeStyle,  
antialiased: Bool = true) ->  
StrokeBorderShapeView<Self.Content, S,  
Self> where S : ShapeStyle  
  
    /// Returns a view that's the result  
    of filling an inner stroke of this view  
    with the content you supply.  
    ///  
    /// This is equivalent to insetting  
    `self` by `lineWidth / 2` and stroking  
    the
```

```
    /// resulting shape with `lineWidth`  
    as the line-width.  
    nonisolated public func  
strokeBorder<S>(_ content: S  
= .foreground, lineWidth: CGFloat = 1,  
antialiased: Bool = true) ->  
StrokeBorderStyleView<Self.Content, S,  
Self> where S : ShapeStyle  
}  
  
/// A gesture containing two gestures  
that can happen at the same time with  
/// neither of them preceding the other.  
///  
/// A simultaneous gesture is a  
container-event handler that evaluates  
its two  
/// child gestures at the same time. Its  
value is a struct with two optional  
/// values, each representing the phases  
of one of the two gestures.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct  
SimultaneousGesture<First, Second> :  
Gesture where First : Gesture, Second :  
Gesture {  
  
    /// The value of a simultaneous  
gesture that indicates which of its two  
    /// gestures receives events.  
    @frozen public struct Value {
```

```
        /// The value of the first
gesture.
    public var first: First.Value?

        /// The value of the second
gesture.
    public var second: Second.Value?
}

/// The first of two gestures that
can happen simultaneously.
public var first: First

/// The second of two gestures that
can happen simultaneously.
public var second: Second

/// Creates a gesture with two
gestures that can receive updates or
succeed
/// independently of each other.
///
/// - Parameters:
///   - first: The first of two
gestures that can happen simultaneously.
///   - second: The second of two
gestures that can happen simultaneously.
@inlinable public init(_ first:
First, _ second: Second)

/// The type of gesture representing
the body of `Self`.
@available(iOS 13.0, tvOS 13.0,
```

```
watchOS 6.0, macOS 10.15, *)
    public typealias Body = Never
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension SimultaneousGesture.Value : Sendable where First.Value : Sendable,
Second.Value : Sendable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension SimultaneousGesture.Value : Equatable where First.Value : Equatable,
Second.Value : Equatable {

    /// 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:
        SimultaneousGesture<First, Second>.Value,
        b: SimultaneousGesture<First,
        Second>.Value) -> Bool
```

```
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension SimultaneousGesture.Value :  
Hashable where First.Value : Hashable,  
Second.Value : Hashable {  
  
    /// 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)

```
    /// 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 }  
  
    /// The size dependent variant preference  
    allows the text to take the available  
    /// space into account when choosing the  
    variant to display.  
    @available(iOS 18.0, macOS 15.0, tvOS  
    18.0, watchOS 11.0, visionOS 2.0, *)  
    public struct SizeDependentTextVariant :  
    TextVariantPreference, Sendable {  
    }  
  
    /// A transition that inserts by moving  
    in from the leading edge, and  
    /// removes by moving out towards the
```

```
trailing edge.  
///  
/// - SeeAlso: `MoveTransition`  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
@MainActor @preconcurrency public struct  
SlideTransition : Transition {  
  
    @MainActor @preconcurrency public  
    init()  
  
        /// Gets the current body of the  
        caller.  
        ///  
        /// `content` is a proxy for the view  
        that will have the modifier  
        /// represented by `Self` applied to  
        it.  
        @MainActor @preconcurrency public  
        func body(content:  
SlideTransition.Content, phase:  
TransitionPhase) -> some View  
  
        /// The type of view representing the  
        body.  
        @available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
        public typealias Body = some View  
    }  
  
    /// A flexible space that expands along  
    the major axis of its containing stack
```

```
/// layout, or on both axes if not
// contained in a stack.
///
/// A spacer creates an adaptive view
// with no content that expands as much as
// it can. For example, when placed
// within an ``HStack``, a spacer expands
// horizontally as much as the stack
// allows, moving sibling views out of the
// way, within the limits of the stack's
// size.
/// SwiftUI sizes a stack that doesn't
// contain a spacer up to the combined
// ideal widths of the content of the
// stack's child views.
///
/// The following example provides a
// simple checklist row to illustrate how
// you
/// can use a spacer:
///
///     struct ChecklistRow: View {
///         let name: String
///
///         var body: some View {
///             HStack {
///                 Image(systemName:
/// "checkmark")
///                 Text(name)
///             }
///             .border(Color.blue)
///         }
///     }
```

```
///  
/// ! [A figure of a blue rectangular  
border that marks the boundary of an  
/// HStack, wrapping a checkmark image to  
the left of the name Megan. The  
/// checkmark and name are centered  
vertically and separated by system  
/// standard-spacing within the stack.]  
(Spacer-1.png)  
///  
/// Adding a spacer before the image  
creates an adaptive view with no content  
/// that expands to push the image and  
text to the right side of the stack.  
/// The stack also now expands to take as  
much space as the parent view allows,  
/// shown by the blue border that  
indicates the boundary of the stack:  
///  
///     struct ChecklistRow: View {  
///         let name: String  
///  
///         var body: some View {  
///             HStack {  
///                 Spacer()  
///                 Image(systemName:  
"checkmark")  
///                     .border(Color.blue)  
///             }  
///         }  
///     }  
///
```

```
/// ! [A figure of a blue rectangular border that marks the boundary of an HStack, wrapping a checkmark image to the left of the name Megan. The checkmark and name are centered vertically, separated by system-standard spacing, and pushed to the right side of the stack.] (Spacer-2.png)
///
/// Moving the spacer between the image and the name pushes those elements to the left and right sides of the ``HStack``, respectively. Because the stack contains the spacer, it expands to take as much horizontal space as the parent view allows; the blue border indicates its size:
///
///     struct ChecklistRow: View {
///         let name: String
///
///         var body: some View {
///             HStack {
///                 Image(systemName: "checkmark")
///                 Spacer()
///                 Text(name)
///             }
///             .border(Color.blue)
///         }
///     }
```

```
/// ! [A figure of a blue rectangular border that marks the boundary of an HStack, wrapping a checkmark image to the left of the name Megan. The checkmark and name are centered vertically, with the checkmark on the left edge of the stack, and the text on the right side of the stack.] (Spacer-3.png)
///
/// Adding two spacer views on the outside of the stack leaves the image and text together, while the stack expands to take as much horizontal space as the parent view allows:
///
///     struct ChecklistRow: View {
///         let name: String
///
///         var body: some View {
///             HStack {
///                 Spacer()
///                 Image(systemName: "checkmark")
///                 Text(name)
///                 Spacer()
///             }
///             .border(Color.blue)
///         }
///     }
///
/// ! [A figure of a blue rectangular border marks the boundary of an HStack,
```

```
/// wrapping a checkmark image to the
left of text spelling the name Megan.
/// The checkmark and name are centered
vertically, separated by
/// system-standard spacing, and centered
horizontally
/// in the stack.] (Spacer-4.png)
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@MainActor @frozen @preconcurrency public
struct Spacer {

    /// The minimum length this spacer
    can be shrunk to, along the axis or axes
    /// of expansion.
    ///
    /// If `nil`, the system default
    spacing between views is used.
    @MainActor @preconcurrency public var
    minLength: CGFloat?

    @MainActor @inlinable @preconcurrency
    public init(minLength: CGFloat? = nil)

    /// The type of view representing the
    body of this view.
    ///
    /// When you create a custom view,
    Swift infers this type from your
    /// implementation of the required
    ``View/body-swift.property`` property.
    @available(iOS 13.0, tvOS 13.0,
    watchOS 6.0, macOS 10.15, *)
```

```
    public typealias Body = Never
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Spacer : View {

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Spacer : Sendable {

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Spacer : BitwiseCopyable {
}

/// A representation of a spring's
/// motion.
///
/// Use this type to convert between
/// different representations of spring
/// parameters:
///
///     let spring = Spring(duration:
/// 0.5, bounce: 0.3)
///     let (mass, stiffness, damping) =
/// (spring.mass, spring.stiffness,
/// spring.damping)
///     // (1.0, 157.9, 17.6)
///
///     let spring2 = Spring(mass: 1,
```

```
stiffness: 100, damping: 10)
///     let (duration, bounce) =
(spring2.duration, spring2.bounce)
///     // (0.63, 0.5)
///
/// You can also use it to query for a
spring's position and its other
properties for
/// a given set of inputs:
///
///     func unitPosition(time:
TimeInterval) -> Double {
///         let spring = Spring(duration:
0.5, bounce: 0.3)
///         return
spring.position(target: 1.0, time: time)
///     }
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct Spring : Hashable, Sendable
{

    /// 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: Spring, b:  
Spring) -> 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 }
}

/// Duration/Bounce Parameters
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Spring {

    /// Creates a spring with the
    specified duration and bounce.
    ///
    /// - Parameters:
    ///   - duration: Defines the pace of
    the spring. This is approximately
    ///   equal to the settling duration,
    but for springs with very large
    ///   bounce values, will be the
    duration of the period of oscillation
    ///   for the spring.
    ///   - bounce: How bouncy the spring
```

```
should be. A value of 0 indicates
    /// no bounces (a critically damped
spring), positive values indicate
    /// increasing amounts of
bounciness up to a maximum of 1.0
    /// (corresponding to undamped
oscillation), and negative values
    /// indicate overdamped springs
with a minimum value of -1.0.
public init(duration: TimeInterval =
0.5, bounce: Double = 0.0)

    /// The perceptual duration, which
defines the pace of the spring.
public var duration: TimeInterval {
get }

    /// How bouncy the spring is.
    ///
    /// A value of 0 indicates no bounces
(a critically damped spring), positive
    /// values indicate increasing
amounts of bounciness up to a maximum of
1.0
    /// (corresponding to undamped
oscillation), and negative values
indicate
    /// overdamped springs with a minimum
value of -1.0.
public var bounce: Double { get }
}

/// Response/DampingRatio Parameters
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Spring {  
  
    /// Creates a spring with the  
    // specified response and damping ratio.  
    ///  
    /// - Parameters:  
    ///     - response: Defines the  
    stiffness of the spring as an approximate  
    ///     duration in seconds.  
    ///     - dampingRatio: Defines the  
    amount of drag applied as a fraction the  
    ///     amount needed to produce  
    critical damping.  
    public init(response: Double,  
                dampingRatio: Double)  
  
        /// The stiffness of the spring,  
        defined as an approximate duration in  
        /// seconds.  
    public var response: Double { get }  
  
        /// The amount of drag applied, as a  
        fraction of the amount needed to  
        /// produce critical damping.  
        ///  
        /// When `dampingRatio` is 1, the  
        spring will smoothly decelerate to its  
        /// final position without  
        oscillating. Damping ratios less than 1  
        will  
        /// oscillate more and more before
```

coming to a complete stop.

```
    public var dampingRatio: Double { get  
}  
}  
}
```

/// Mass/Stiffness/Damping Parameters

@available(iOS 17.0, macOS 14.0, tvOS

17.0, watchOS 10.0, \*)

extension Spring {

/// Creates a spring with the  
 specified mass, stiffness, and damping.

///

/// - Parameters:

/// - mass: Specifies that property  
 of the object attached to the end of

/// the spring.

/// - stiffness: The corresponding  
 spring coefficient.

/// - damping: Defines how the  
 spring's motion should be damped due to  
 the

/// forces of friction.

/// - allowOverdamping: A value of  
 true specifies that over-damping

/// should be allowed when  
 appropriate based on the other inputs,  
 and a

/// value of false specifies that  
 such cases should instead be treated as

/// critically damped.

```
    public init(mass: Double = 1.0,  
                stiffness: Double, damping: Double,
```

```
allowOverDamping: Bool = false)

    /// The mass of the object attached
    to the end of the spring.
    ///
    /// The default mass is 1. Increasing
    this value will increase the spring's
    /// effect: the attached object will
    be subject to more oscillations and
    /// greater overshoot, resulting in
    an increased settling duration.
    /// Decreasing the mass will reduce
    the spring effect: there will be fewer
    /// oscillations and a reduced
    overshoot, resulting in a decreased
    /// settling duration.
public var mass: Double { get }

    /// The spring stiffness coefficient.
    ///
    /// Increasing the stiffness reduces
    the number of oscillations and will
    /// reduce the settling duration.
    Decreasing the stiffness increases the
    the
    /// number of oscillations and will
    increase the settling duration.
public var stiffness: Double { get }

    /// Defines how the spring's motion
    should be damped due to the forces of
    /// friction.
    ///
```

```
    /// Reducing this value reduces the
    energy loss with each oscillation: the
    /// spring will overshoot its
destination. Increasing the value
increases
    /// the energy loss with each
duration: there will be fewer and smaller
    /// oscillations.
public var damping: Double { get }
}

/// SettlingDuration/DampingRatio
Parameters
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Spring {

    /// Creates a spring with the
specified duration and damping ratio.
    ///
    /// - Parameters:
    ///   - settlingDuration: The
approximate time it will take for the
spring
    ///   to come to rest.
    ///   - dampingRatio: The amount of
drag applied as a fraction of the amount
    ///   needed to produce critical
damping.
    ///   - epsilon: The threshold for
how small all subsequent values need to
    ///   be before the spring is
considered to have settled.
```

```
    public init(settlingDuration:  
TimeInterval, dampingRatio: Double,  
epsilon: Double = 0.001)  
}  
  
/// VectorArithmetic Evaluation  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Spring {  
  
    /// The estimated duration required  
    /// for the spring system to be considered  
    /// at rest.  
    ///  
    /// This uses a `target` of 1.0, an  
    /// `initialVelocity` of 0, and an `epsilon`  
    /// of 0.001.  
    public var settlingDuration:  
TimeInterval { get }  
  
    /// The estimated duration required  
    /// for the spring system to be considered  
    /// at rest.  
    ///  
    /// The epsilon value specifies the  
    /// threshhold for how small all subsequent  
    /// values need to be before the  
    /// spring is considered to have settled.  
    public func  
settlingDuration<V>(target: V,  
initialVelocity: V = .zero, epsilon:  
Double) -> TimeInterval where V :  
VectorArithmetic
```

```
    /// Calculates the value of the
    spring at a given time given a target
    /// amount of change.
    public func value<V>(target: V,
initialVelocity: V = .zero, time:
TimeInterval) -> V where V :
VectorArithmetic

    /// Calculates the velocity of the
    spring at a given time given a target
    /// amount of change.
    public func velocity<V>(target: V,
initialVelocity: V = .zero, time:
TimeInterval) -> V where V :
VectorArithmetic

    /// Updates the current value and
velocity of a spring.
    ///
    /// - Parameters:
    ///   - value: The current value of
the spring.
    ///   - velocity: The current
velocity of the spring.
    ///   - target: The target that
`value` is moving towards.
    ///   - deltaTime: The amount of time
that has passed since the spring was
    ///       at the position specified by
`value`.
    public func update<V>(value: inout V,
velocity: inout V, target: V, deltaTime:
```

```
TimeInterval) where V : VectorArithmetic

    /// Calculates the force upon the
    spring given a current position, target,
    /// and velocity amount of change.
    ///
    /// This value is in units of the
    vector type per second squared.
    public func force<V>(target: V,
position: V, velocity: V) -> V where V :
VectorArithmetic
}

/// Animatable Evaluation
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Spring {

    /// The estimated duration required
    for the spring system to be considered
    /// at rest.
    ///
    /// The epsilon value specifies the
    threshhold for how small all subsequent
    /// values need to be before the
    spring is considered to have settled.
    public func
settlingDuration<V>(fromValue: V,
toValue: V, initialVelocity: V, epsilon:
Double) -> TimeInterval where V :
Animatable

    /// Calculates the value of the
```

```
spring at a given time for a starting
    /// and ending value for the spring
to travel.
    public func value<V>(fromValue: V,
toValue: V, initialVelocity: V, time:
TimeInterval) -> V where V : Animatable

    /// Calculates the velocity of the
spring at a given time given a starting
    /// and ending value for the spring
to travel.
    public func velocity<V>(fromValue: V,
toValue: V, initialVelocity: V, time:
TimeInterval) -> V where V : Animatable

    /// Calculates the force upon the
spring given a current position,
    /// velocity, and divisor from the
starting and end values for the spring to
travel.
    /**
     /// This value is in units of the
vector type per second squared.
    public func force<V>(fromValue: V,
toValue: V, position: V, velocity: V) ->
V where V : Animatable
}

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

    /// A smooth spring with a predefined
```

duration and no bounce.

```
    public static var smooth: Spring {  
        get }
```

/// A smooth spring with a predefined duration and no bounce that can be

/// tuned.

///

/// - Parameters:

/// - duration: The perceptual duration, which defines the pace of the

/// spring. This is approximately equal to the settling duration, but

/// for very bouncy springs, will be the duration of the period of

/// oscillation for the spring.

/// - extraBounce: How much additional bounce should be added to the base

/// bounce of 0.

```
    public static func smooth(duration:  
        TimeInterval = 0.5, extraBounce: Double =  
        0.0) -> Spring
```

/// A spring with a predefined duration and small amount of bounce that

/// feels more snappy.

```
    public static var snappy: Spring {  
        get }
```

/// A spring with a predefined duration and small amount of bounce that

/// feels more snappy and can be

tuned.

```
///
/// - Parameters:
///   - duration: The perceptual
duration, which defines the pace of the
///   spring. This is approximately
equal to the settling duration, but
///   for very bouncy springs, will
be the duration of the period of
///   oscillation for the spring.
///   - extraBounce: How much
additional bounciness should be added to
the
///   base bounce of 0.15.
public static func snappy(duration:
TimeInterval = 0.5, extraBounce: Double =
0.0) -> Spring

    /// A spring with a predefined
duration and higher amount of bounce.
public static var bouncy: Spring {
get }

    /// A spring with a predefined
duration and higher amount of bounce that
/// can be tuned.
///
/// - Parameters:
///   - duration: The perceptual
duration, which defines the pace of the
///   spring. This is approximately
equal to the settling duration, but
///   for very bouncy springs, will
```

```
be the duration of the period of
    ///      oscillation for the spring.
    /// - extraBounce: How much
additional bounce should be added to the
base
    ///      bounce of 0.3.
    /// - blendDuration: The duration
in seconds over which to interpolate
    ///      changes to the duration.
    public static func bouncy(duration:
TimeInterval = 0.5, extraBounce: Double =
0.0) -> Spring
}
```

```
/// A keyframe that uses a spring
function to interpolate to the given
value.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct SpringKeyframe<Value> :
KeyframeTrackContent where Value :
Animatable {

    /// Creates a new keyframe using the
given value and timestamp.
    ///
    /// - Parameters:
    /// - to: The value of the
keyframe.
    /// - duration: The duration of the
segment defined by this keyframe,
    /// or nil to use the settling
duration of the spring.
```

```
    /// - spring: The spring that
    defines the shape of the segment before
    /// this keyframe
    /// - startVelocity: The velocity
    of the value at the start of the
    /// segment, or `nil` to
    automatically compute the velocity to
    maintain
    /// smooth motion.
public init(_ to: Value, duration:
TimeInterval? = nil, spring: Spring =
Spring(), startVelocity: Value? = nil)

    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Body =
SpringKeyframe<Value>
}

/// A property wrapper type that can read
and write a value managed by SwiftUI.
///
/// Use state as the single source of
truth for a given value type that you
/// store in a view hierarchy. Create a
state value in an ``App``, ``Scene``,
/// or ``View`` by applying the `@State`
attribute to a property declaration
/// and providing an initial value.
Declare state as private to prevent
setting
/// it in a memberwise initializer, which
can conflict with the storage
```

```
/// management that SwiftUI provides:  
///  
///     struct PlayButton: View {  
///         @State private var.isPlaying:  
Bool = false // Create the state.  
///  
///         var body: some View {  
///             Button(isPlaying ?  
"Pause" : "Play") { // Read the state.  
///                 isPlaying.toggle() //  
Write the state.  
///             }  
///         }  
///     }  
///  
/// SwiftUI manages the property's  
storage. When the value changes, SwiftUI  
/// updates the parts of the view  
hierarchy that depend on the value.  
/// To access a state's underlying value,  
you use its ``wrappedValue`` property.  
/// However, as a shortcut Swift enables  
you to access the wrapped value by  
/// referring directly to the state  
instance. The above example reads and  
/// writes the `isPlaying` state  
property's wrapped value by referring to  
the  
/// property directly.  
///  
/// Declare state as private in the  
highest view in the view hierarchy that  
/// needs access to the value. Then share
```

the state with any subviews that also  
/// need access, either directly for  
read-only access, or as a binding for  
/// read-write access. You can safely  
mutate state properties from any thread.  
///  
/// **Share state with subviews**  
///  
/// If you pass a state property to a  
Subview, SwiftUI updates the subview  
/// any time the value changes in the  
container view, but the subview can't  
/// modify the value. To enable the  
Subview to modify the state's stored  
value,  
/// pass a ``Binding`` instead.  
///  
/// For example, you can remove the  
'.isPlaying` state from the play button in  
/// the above example, and instead make  
the button take a binding:  
///  
/// struct PlayButton: View {  
/// @Binding var isPlaying:  
Bool // Play button now receives a  
binding.  
///  
/// var body: some View {  
/// Button(isPlaying ?  
"Pause" : "Play") {  
/// isPlaying.toggle()  
/// }  
/// }  
/// }

```
///      }
///
/// Then you can define a player view
/// that declares the state and creates a
/// binding to the state. Get the binding
/// to the state value by accessing the
/// state's ``projectedValue``, which you
/// get by prefixing the property name
/// with a dollar sign (`$`):
///
///     struct PlayerView: View {
///         @State private var.isPlaying:
Bool = false // Create the state here
now.
///
///         var body: some View {
///             VStack {
///                 PlayButton(isPlaying:
$isPlaying) // Pass a binding.
///
///                     // ...
///             }
///         }
///
/// Like you do for a ``StateObject``,
declare `State` as private to prevent
/// setting it in a memberwise
initializer, which can conflict with the
storage
/// management that SwiftUI provides.
Unlike a state object, always
/// initialize state by providing a
```

```
default value in the state's
/// declaration, as in the above
examples. Use state only for storage
that's
/// local to a view and its subviews.
///
/// ### Store observable objects
///
/// You can also store observable objects
that you create with the
///
<doc://com.apple.documentation/documentation/Observation/Observable()
/// macro in `State`; for example:
///
///     @Observable
///     class Library {
///         var name = "My library of
books"
///         // ...
///     }
///
///     struct ContentView: View {
///         @State private var library =
Library()
///
///         var body: some View {
///             LibraryView(library:
library)
///         }
///     }
///
/// A `State` property always
```

instantiates its default value when SwiftUI

```
/// instantiates the view. For this reason, avoid side effects and
/// performance-intensive work when initializing the default value. For
/// example, if a view updates frequently, allocating a new default object each
/// time the view initializes can become expensive. Instead, you can defer the
/// creation of the object using the ``View/task(priority:_:)`` modifier,
which
/// is called only once when the view first appears:
```

```
///
///     struct ContentView: View {
///         @State private var library:
Library?
///
///         var body: some View {
///             LibraryView(library:
library)
///             .task {
///                 library =
Library()
///
///             }
///         }
///     }
```

/// Delaying the creation of the observable state object ensures that

```
/// unnecessary allocations of the object  
/// doesn't happen each time SwiftUI  
/// initializes the view. Using the  
``View/task(priority:_:)`` modifier is  
also  
/// an effective way to defer any other  
kind of work required to create the  
/// initial state of the view, such as  
network calls or file access.  
///  
/// > Note: It's possible to store an  
object that conforms to the  
///  
<doc://com.apple.documentation/documentation/Combine/ObservableObject>  
/// protocol in a `State` property.  
However the view will only update when  
/// the reference to the object changes,  
such as when setting the property with  
/// a reference to another object. The  
view will not update if any of the  
/// object's published properties change.  
To track changes to both the reference  
/// and the object's published  
properties, use ``StateObject`` instead  
of  
/// ``State`` when storing the object.  
///  
/// ### Share observable state objects  
with subviews  
///  
/// To share an  
<doc://com.apple.documentation/documentation/combine/ObservableObject>
```

```
ion/Observation/Observable>
/// object stored in `State` with a
subview, pass the object reference to
/// the subview. SwiftUI updates the
subview anytime an observable property of
/// the object changes, but only when the
subview's ``View/body`` reads the
/// property. For example, in the
following code `BookView` updates each
time
/// `title` changes but not when
`isAvailable` changes:
///
///     @Observable
///     class Book {
///         var title = "A sample book"
///         var isAvailable = true
///     }

///
///     struct ContentView: View {
///         @State private var book =
Book()
///

///         var body: some View {
///             BookView(book: book)
///         }
///     }

///
///     struct BookView: View {
///         var book: Book
///

///         var body: some View {
///             Text(book.title)
```

```
///      }
///      }
///
/// `State` properties provide bindings
/// to their value. When storing an object,
/// you can get a ``Binding`` to that
/// object, specifically the reference to the
/// object. This is useful when you need
/// to change the reference stored in
/// state in some other subview, such as
/// setting the reference to `nil`:
///
/// struct ContentView: View {
///     @State private var book:
Book?
///
///     var body: some View {
///         DeleteBookView(book:
$book)
///
///             .task {
///                 book = Book()
///             }
///         }
///     }
///
/// struct DeleteBookView: View {
///     @Binding var book: Book?
///
///     var body: some View {
///         Button("Delete book") {
///             book = nil
///         }
///     }
/// }
```

```
///      }
///
/// However, passing a ``Binding`` to an
object stored in `State` isn't
/// necessary when you need to change
properties of that object. For example,
/// you can set the properties of the
object to new values in a subview by
/// passing the object reference instead
of a binding to the reference:
///
///      struct ContentView: View {
///          @State private var book =
Book()
///
///          var body: some View {
///              BookCheckoutView(book:
book)
///          }
///      }
///
///      struct BookCheckoutView: View {
///          var book: Book
///
///          var body: some View {
///              Button(book.isAvailable ?
"Check out book" : "Return book") {
///                  book.isAvailable.toggle()
///              }
///          }
///      }
///
```

```
/// If you need a binding to a specific
/// property of the object, pass either the
/// binding to the object and extract
bindings to specific properties where
/// needed, or pass the object reference
and use the ``Bindable`` property
/// wrapper to create bindings to
specific properties. For example, in the
/// following code `BookEditorView` wraps
`book` with `@Bindable`. Then the
/// view uses the `$` syntax to pass to a
``TextField`` a binding to `title`:
///
///     struct ContentView: View {
///         @State private var book =
Book()
///
///         var body: some View {
///             BookView(book: book)
///         }
///     }
///
///     struct BookView: View {
///         let book: Book
///
///         var body: some View {
///             BookEditorView(book:
book)
///         }
///     }
///
///     struct BookEditorView: View {
///         @Bindable var book: Book
```

```
///  
///         var body: some View {  
///             TextField("Title", text:  
$book.title)  
///         }  
///     }  
///  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen @propertyWrapper public struct  
State<Value> : DynamicProperty {  
  
    /// Creates a state property that  
    stores an initial wrapped value.  
    ///  
    /// You don't call this initializer  
    directly. Instead, SwiftUI  
    /// calls it for you when you declare  
    a property with the `@State`  
    /// attribute and provide an initial  
    value:  
    ///  
    ///     struct MyView: View {  
    ///         @State private var  
   .isPlaying: Bool = false  
    ///  
    ///         // ...  
    ///     }  
    ///  
    ///     /// SwiftUI initializes the state's  
    storage only once for each  
    /// container instance that you  
    declare. In the above code, SwiftUI
```

```
    /// creates `isPlaying` only the
first time it initializes a particular
    /// instance of `MyView`. On the
other hand, each instance of `MyView`
    /// creates a distinct instance of
the state. For example, each of
    /// the views in the following
``VStack`` has its own `isPlaying` value:
```

```
    ///
    ///     var body: some View {
    ///         VStack {
    ///             MyView()
    ///             MyView()
    ///         }
    ///     }
    ///
    /// - Parameter value: An initial
value to store in the state
    /// property.
```

```
public init(wrappedValue value:
Value)
```

```
    /// Creates a state property that
stores an initial value.
    ///
    /// This initializer has the same
behavior as the ``init(wrappedValue:)``
    /// initializer. See that initializer
for more information.
    ///
    /// - Parameter value: An initial
value to store in the state
    /// property.
```

```
    public init(initialValue value:  
Value)  
  
        /// The underlying value referenced  
by the state variable.  
        ///  
        /// This property provides primary  
access to the value's data. However, you  
        /// don't typically access  
`wrappedValue` explicitly. Instead, you  
gain  
        /// access to the wrapped value by  
referring to the property variable that  
        /// you create with the `@State`  
attribute.  
        ///  
        /// In the following example, the  
button's label depends on the value of  
        /// `isPlaying` and the button's  
action toggles the value of `isPlaying`.  
        /// Both of these accesses implicitly  
access the state property's wrapped  
        /// value:  
        ///  
        ///     struct PlayButton: View {  
        ///         @State private var  
isPlaying: Bool = false  
        ///  
        ///             var body: some View {  
        ///                 Button(isPlaying ?  
"Pause" : "Play") {  
        ///                     isPlaying.toggle()  
        ///                 }  
        ///             }  
        ///         }  
        ///     }  
        /// }  
    }  
}
```

```
    /// }  
    /// }  
    /// }  
    /// }  
    public var wrappedValue: Value { get  
nonmutating set }
```

```
    /// A binding to the state value.  
    ///  
    /// Use the projected value to get a  
``Binding`` to the stored value. The  
    /// binding provides a two-way  
connection to the stored value. To access  
    /// the `projectedValue`, prefix the  
property variable with a dollar  
    /// sign (`$`).  
    ///  
    /// In the following example,  
`PlayerView` projects a binding of the  
state  
    /// property `isPlaying` to the  
`PlayButton` view using `.isPlaying`. That
```

```
    /// enables the play button to both
read and write the value:
    ///
    /// struct PlayerView: View {
    ///     var episode: Episode
    ///     @State private var
isPlaying: Bool = false
    ///
    ///         var body: some View {
    ///             VStack {
```

```
    /**
     Text(episode.title)
        /**
         .foregroundStyle(isPlaying ? .primary : .secondary)
        /**
     PlayButton(isPlaying: $isPlaying)
        /**
         }
        /**
         }
        /**
         */
        public var projectedValue:
Binding<Value> { get }
}

@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension State : Sendable where Value : Sendable {
}

@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension State where Value : ExpressibleByNilLiteral {

    /**
     Creates a state property without an initial value.
    /**
     /**
      This initializer behaves like the ``init(wrappedValue:)`` initializer
      /**
       with an input of `nil`. See that initializer for more information.
    @inlinable public init()
```

```
}
```

```
/// A property wrapper type that  
instantiates an observable object.  
///  
/// Use a state object as the single  
source of truth for a reference type that  
/// you store in a view hierarchy. Create  
a state object in an ``App``,  
/// ``Scene``, or ``View`` by applying  
the `@StateObject` attribute to a  
/// property declaration and providing an  
initial value that conforms to the  
///  
 <doc://com.apple.documentation/documentation/Combine/ObservableObject>  
/// protocol. Declare state objects as  
private to prevent setting them from a  
/// memberwise initializer, which can  
conflict with the storage management that  
/// SwiftUI provides:  
///  
///     class DataModel: ObservableObject  
{  
///         @Published var name = "Some  
Name"  
///         @Published var isEnabled =  
false  
///     }  
///  
///     struct MyView: View {  
///         @StateObject private var  
model = DataModel() // Create the state
```

```
object.  
///  
///         var body: some View {  
///             Text(model.name) //  
Updates when the data model changes.  
///             MySubView()  
///             .environmentObject(mo  
del)  
///         }  
///     }  
///  
/// SwiftUI creates a new instance of the  
model object only once during the  
/// lifetime of the container that  
declares the state object. For example,  
/// SwiftUI doesn't create a new instance  
if a view's inputs change, but does  
/// create a new instance if the identity  
of a view changes. When published  
/// properties of the observable object  
change, SwiftUI updates any view that  
/// depends on those properties, like the  
``Text`` view in the above example.  
///  
/// > Note: If you need to store a value  
type, like a structure, string, or  
/// integer, use the ``State`` property  
wrapper instead. Also use ``State``  
/// if you need to store a reference  
type that conforms to the  
///  
<doc://com.apple.documentation/documentation/Observation/Observable()%gt;
```

```
/// protocol. To learn more about
Observation in SwiftUI, see
/// <doc:Managing-model-data-in-your-
app>.
///
/// Share state objects with subviews
///
/// You can pass a state object into a
subview through a property that has the
/// ``ObservedObject`` attribute.
Alternatively, add the object to the
/// environment of a view hierarchy by
applying the
/// ``View/environmentObject(_:)``
modifier to a view, like `MySubView` in
the
/// above code. You can then read the
object inside `MySubView` or any of its
/// descendants using the
``EnvironmentObject`` attribute:
///
///     struct MySubView: View {
///         @EnvironmentObject var model:
DataModel
///
///         var body: some View {
///             Toggle("Enabled", isOn:
$model.isEnabled)
///         }
///     }
///
/// Get a ``Binding`` to the state
object's properties using the dollar sign
```

```
/// (`$`) operator. Use a binding when
/// you want to create a two-way connection.
/// In the above code, the ``Toggle``
/// controls the model's `isEnabled` value
/// through a binding.
///
/// ### Initialize state objects using
external data
///
/// When a state object's initial state
depends on data that comes from
/// outside its container, you can call
the object's initializer
/// explicitly from within its
container's initializer. For example,
/// suppose the data model from the
previous example takes a `name`
/// input during initialization and you
want to use a value for that
/// name that comes from outside the
view. You can do this with
/// a call to the state object's
initializer inside an explicit
initializer
/// that you create for the view:
///
///     struct MyInitializableView: View
{
///         @StateObject private var
model: DataModel
///
///         init(name: String) {
///             // SwiftUI ensures that
```

the following initialization uses the

```
///           // closure only once
during the lifetime of the view, so
///           // later changes to the
view's name input have no effect.
///           _model =
StateObject(wrappedValue: DataModel(name:
name))
///           }
///
///
var body: some View {
    VStack {
        Text("Name: \
(model.name)")
    }
}
///
///
/// Use caution when doing this. SwiftUI
only initializes a state object
/// the first time you call its
initializer in a given view. This
/// ensures that the object provides
stable storage even as the view's
/// inputs change. However, it might
result in unexpected behavior or
/// unwanted side effects if you
explicitly initialize the state object.
///
/// In the above example, if the `name`
input to `MyInitializableView`
/// changes, SwiftUI reruns the view's
initializer with the new value. However,
```

```
/// SwiftUI runs the autoclosure that you
/// provide to the state object's
/// initializer only the first time you
/// call the state object's initializer, so
/// the model's stored `name` value
/// doesn't change.
///
/// Explicit state object initialization
/// works well when the external data
/// that the object depends on doesn't
/// change for a given instance of the
/// object's container. For example, you
/// can create two views with different
/// constant names:
///
///     var body: some View {
///         VStack {
///             MyInitializableView(name:
/// "Ravi")
///             MyInitializableView(name:
/// "Maria")
///         }
///     }
///
/// > Important: Even for a configurable
/// state object, you still declare it
/// as private. This ensures that you
/// can't accidentally set the parameter
/// through a memberwise initializer of
/// the view, because doing so can
/// conflict with the framework's
/// storage management and produce unexpected
/// results.
```

```
///  
/// Force reinitialization by changing view identity  
///  
/// If you want SwiftUI to reinitialize a state object when a view input  
/// changes, make sure that the view's identity changes at the same time.  
/// One way to do this is to bind the view's identity to the value that changes  
/// using the ``View/id(_:)`` modifier.  
For example, you can ensure that  
/// the identity of an instance of `MyInitializableView` changes when its  
/// `name` input changes:  
///  
///     MyInitializableView(name: name)  
///         .id(name) // Binds the  
identity of the view to the name  
property.  
///  
/// > NOTE: If your view appears inside a ``ForEach``, it implicitly receives an  
/// ``View/id(_:)`` modifier that uses the identifier of the corresponding  
/// data element.  
///  
/// If you need the view to reinitialize state based on changes in more than  
/// one value, you can combine the values into a single identifier using a  
///  
<doc://com.apple.documentation/documentat
```

ion/Swift/Hasher>. For example,

```
/// if you want to update the data model  
in `MyInitializableView` when the  
/// values of either `name` or  
`isEnabled` change, you can combine both  
/// variables into a single hash:  
///  
///     var hash: Int {  
///         var hasher = Hasher()  
///         hasher.combine(name)  
///         hasher.combine(isEnabled)  
///         return hasher.finalize()  
///     }  
///  
/// Then apply the combined hash to the  
view as an identifier:  
///  
///     MyInitializableView(name: name,  
isEnabled: isEnabled)  
///         .id(hash)  
///  
/// Be mindful of the performance cost of  
reinitializing the state object every  
/// time the input changes. Also,  
changing view identity can have side  
/// effects. For example, SwiftUI doesn't  
automatically animate  
/// changes inside the view if the view's  
identity changes at the same time.  
/// Also, changing the identity resets  
all state held by the view, including  
/// values that you manage as ``State``,  
``FocusState``, ``GestureState``,
```

```
/// and so on.  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
@MainActor @frozen @propertyWrapper  
@preconcurrency public struct  
StateObject<ObjectType> : DynamicProperty  
where ObjectType : ObservableObject {  
  
    /// Creates a new state object with  
    an initial wrapped value.  
    ///  
    /// You typically don't call this  
    initializer directly. Instead, SwiftUI  
    /// calls it for you when you declare  
    a property with the `@StateObject`  
    /// attribute in an ``App``,  
    ``Scene``, or ``View`` and provide an  
    initial  
    /// value:  
    ///  
    ///     struct MyView: View {  
    ///         @StateObject private var  
model = DataModel()  
    ///  
    ///         // ...  
    ///     }  
    ///  
    ///     SwiftUI creates only one instance  
    of the state object for each  
    /// container instance that you  
    declare. In the above code, SwiftUI  
    /// creates `model` only the first  
    time it initializes a particular
```

```
    /// instance of `MyView`. On the
    other hand, each instance of `MyView`
    /// creates a distinct instance of
    the data model. For example, each of
    /// the views in the following
``VStack`` has its own model storage:
    ///
    ///     var body: some View {
    ///         VStack {
    ///             MyView()
    ///             MyView()
    ///         }
    ///     }
    ///
    /// /// ### Initialize using external
data
    ///
    /// If the initial state of a state
    object depends on external data, you can
    /// call this initializer directly.
    However, use caution when doing this,
    /// because SwiftUI only initializes
    the object once during the lifetime of
    /// the view --- even if you call the
    state object initializer more than
    /// once --- which might result in
    unexpected behavior. For more information
    /// and an example, see
``StateObject``.
    ///
    /// - Parameter thunk: An initial
    value for the state object.
@inlinable nonisolated public
```

```
init(wrappedValue thunk: @autoclosure  
@escaping () -> ObjectType)  
  
    /// The underlying value referenced  
    by the state object.  
    ///  
    /// The wrapped value property  
    provides primary access to the value's  
    data.  
    /// However, you don't typically  
    access it directly. Instead,  
    /// SwiftUI accesses this property  
    for you when you refer to the variable  
    /// that you create with the  
`@StateObject` attribute:  
    ///  
    ///     @StateObject private var  
contact = Contact()  
    ///  
    ///     var body: some View {  
    ///         Text(contact.name) //  
    Reads name from contact's wrapped value.  
    ///     }  
    ///  
    /// When you change a wrapped value,  
    you can access the new  
    /// value immediately. However,  
    SwiftUI updates views that display the  
    value  
    /// asynchronously, so the interface  
    might not update immediately.  
    @MainActor @preconcurrency public var  
wrappedValue: ObjectType { get }
```

```
    /// A projection of the state object
    /// that creates bindings to its
    /// properties.
    ///
    /// Use the projected value to get a
    ``Binding`` to a property of a state
    /// object. To access the projected
    value, prefix the property name
    /// with a dollar sign (`$`). For
    example, you can get a binding to a
    /// model's `isEnabled` Boolean so
    that a ``Toggle`` can control the value:
    ///
    ///     struct MyView: View {
    ///         @StateObject private var
model = DataModel()
    ///
    ///             var body: some View {
    ///                 Toggle("Enabled",
isOn: $model.isEnabled)
    ///             }
    ///         }
    ///
    /// > Important: A `Binding` created
    by the projected value must only be
    /// read from, or written to by the
    main actor. Failing to do so may result
    /// in undefined behavior, or data
    loss. When this occurs, SwiftUI will
    /// issue a runtime warning. In a
    future release, a crash will occur
    /// instead.
```

```
    @MainActor @preconcurrency public var
projectedValue: ObservedObject<ObjectType>.Wrapper {
get }
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension StateObject : Sendable {

}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension StateObject.Storage : Sendable {
}

/// A shape provider that strokes the
border of its shape.
///
/// You don't create this type directly;
it's the return type of
/// `Shape.strokeBorder`.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@frozen public struct
StrokeBorderShapeView<Content, Style,
Background> : ShapeView where Content :
InsettableShape, Style : ShapeStyle,
Background : View {

    /// The shape that this type draws
and provides for other drawing
```

```
    /// operations.  
    public var shape: Content  
  
    /// The style that strokes the border  
    of this view's shape.  
    public var style: Style  
  
    /// The stroke style used when  
    stroking this view's shape.  
    public var strokeStyle: StrokeStyle  
  
    /// Whether this shape should be  
    drawn antialiased.  
    public var isAntialiased: Bool  
  
    /// The background shown beneath this  
    view.  
    public var background: Background  
  
    /// Create a stroke border shape.  
    public init(shape: Content, style:  
               Style, strokeStyle: StrokeStyle,  
               isAntialiased: Bool, background:  
               Background)  
  
    /// The type of view representing the  
    body of this view.  
    ///  
    /// When you create a custom view,  
    Swift infers this type from your  
    /// implementation of the required  
    ``View/body-swift.property`` property.  
    @available(iOS 17.0, tvOS 17.0,
```

```
watchOS 10.0, macOS 14.0, *)
    public typealias Body = Never
}

/// A shape provider that strokes its
shape.
///
/// You don't create this type directly;
it's the return type of
/// `Shape.stroke`.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@frozen public struct
StrokeShapeView<Content, Style,
Background> : ShapeView where Content :
Shape, Style : ShapeStyle, Background :
View {

    /// The shape that this type draws
and provides for other drawing
    /// operations.
    public var shape: Content

    /// The style that strokes this
view's shape.
    public var style: Style

    /// The stroke style used when
stroking this view's shape.
    public var strokeStyle: StrokeStyle

    /// Whether this shape should be
drawn antialiased.
```

```
public var isAntialiased: Bool

    /// The background shown beneath this
view.
public var background: Background

    /// Create a StrokeShapeView.
public init(shape: Content, style:
Style, strokeStyle: StrokeStyle,
isAntialiased: Bool, background:
Background)

    /// The type of view representing the
body of this view.
    /**
     * When you create a custom view,
Swift infers this type from your
     * implementation of the required
``View/body-swift.property`` property.
@available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
public typealias Body = Never
}

/// The characteristics of a stroke that
traces a path.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct StrokeStyle :
Equatable {

    /// The width of the stroked path.
public var lineWidth: CGFloat
```

```
    /// The endpoint style of a line.
    public var lineCap: CGLineCap

    /// The join type of a line.
    public var lineJoin: CGLineJoin

    /// A threshold used to determine
    whether to use a bevel instead of a
    /// miter at a join.
    public var miterLimit: CGFloat

    /// The lengths of painted and
    unpainted segments used to make a dashed
    line.
    public var dash: [CGFloat]

    /// How far into the dash pattern the
    line starts.
    public var dashPhase: CGFloat

    /// Creates a new stroke style from
    the given components.
    ///
    /// - Parameters:
    ///   - lineWidth: The width of the
    segment.
    ///   - lineCap: The endpoint style
    of a segment.
    ///   - lineJoin: The join type of a
    segment.
    ///   - miterLimit: The threshold
    used to determine whether to use a bevel
```

```
    ///      instead of a miter at a join.  
    /// - dash: The lengths of painted  
and unpainted segments used to make a  
    ///      dashed line.  
    /// - dashPhase: How far into the  
dash pattern the line starts.  
    public init(lineWidth: CGFloat = 1,  
lineCap: CGLineCap = .butt, lineJoin:  
CGLineJoin = .miter, miterLimit: CGFloat  
= 10, dash: [CGFloat] = [CGFloat](),  
dashPhase: CGFloat = 0)  
  
        /// 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:  
StrokeStyle, b: StrokeStyle) -> Bool  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension StrokeStyle : Animatable {  
  
    /// The type defining the data to
```

```
animate.  
    public typealias AnimatableData =  
        AnimatablePair<CGFloat,  
        AnimatablePair<CGFloat, CGFloat>>  
  
        /// The data to animate.  
    public var animatableData:  
        StrokeStyle.AnimatableData  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension StrokeStyle : Sendable {  
}  
  
/// An opaque value representing a  
subview of another view.  
///  
/// Access to a `Subview` can be obtained  
by using `ForEach(subviews:)` or  
/// `Group(subviews:)`.  
///  
/// Subviews are proxies to the resolved  
view they represent, meaning  
/// that modifiers applied to the  
original view will be applied before  
/// modifiers applied to the subview, and  
the view is resolved  
/// using the environment of its  
container, *not* the environment of the  
/// its subview proxy. Additionally,  
because subviews must represent a  
/// single leaf view, or container, a
```

```
Subview may represent a view after the
/// application of styles. As such,
attempting to apply a style to it may
/// have no affect.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct Subview : View,
Identifiable {

    /// A unique identifier for a
    subview.
    public struct ID : Hashable {

        /// 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:  
Subview.ID, b: Subview.ID) -> 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 }
```

/// The unique identifier of the view.

///

/// This identifier persists across updates, changes to the order of

/// subviews, etc. so can be used to track the lifetime of a subview.

```
public var id: Subview.ID { get }
```

/// The container values associated with the given subview.

```
public var containerValues:  
ContainerValues { get }
```

/// The type of view representing the body of this view.

///

/// When you create a custom view, Swift infers this type from your

/// implementation of the required

```
``View/body-swift.property`` property.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias Body = Never  
}  
  
/// An opaque collection representing the  
subviews of view.  
///  
/// Subviews collection constructs  
subviews on demand, so only access the  
part  
/// of the collection you need to create  
the resulting content.  
///  
/// You can get access to a view's  
Subview collection by using the  
/// ``Group/init(sectionsOf:transform:)``  
initializer.  
///  
/// The collection's elements are the  
pieces that make up the given view, and  
/// the collection as a whole acts as a  
proxy for the original view.  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
public struct SubviewsCollection :  
RandomAccessCollection {  
  
    /// Returns the position immediately  
before the given index.  
    ///
```

```
    /// - Parameter i: A valid index of  
the collection. `i` must be greater than  
    /// `startIndex`.  
    /// - Returns: The index value  
immediately before `i`.  
    public func index(before i: Int) ->  
Int
```

```
    /// Returns the position immediately  
after the given index.  
    ///  
    /// The successor of an index must be  
well defined. For an index `i` into a  
    /// collection `c`, calling  
`c.index(after: i)` returns the same  
index every  
    /// time.  
    ///  
    /// - Parameter i: A valid index of  
the collection. `i` must be less than  
    /// `endIndex`.  
    /// - Returns: The index value  
immediately after `i`.  
    public func index(after i: Int) ->  
Int
```

```
    /// Accesses the element at the  
specified position.  
    ///  
    /// The following example accesses an  
element of an array through its  
    /// subscript to print its value:  
    ///
```

```
    ///      var streets = ["Adams",
"Yankee", "Channing", "Douglas",
"Evarts"]
    ///      print(streets[1])
    ///      // Prints "Yankee"
    ///
    /// You can subscript a collection
with any valid index other than the
    /// collection's end index. The end
index refers to the position one past
    /// the last element of a collection,
so it doesn't correspond with an
    /// element.
    ///
    /// - Parameter position: The
position of the element to access.
`position`
    ///      must be a valid index of the
collection that is not equal to the
    ///      `endIndex` property.
    ///
    /// - Complexity: O(1)
public subscript(index: Int) ->
Subview { get }

    /// Accesses a contiguous subrange of
the collection's elements.
    ///
    /// The accessed slice uses the same
indices for the same elements as the
    /// original collection uses. Always
use the slice's `startIndex` property
    /// instead of assuming that its
```

indices start at a particular value.

```
///
/// This example demonstrates getting
a slice of an array of strings, finding
/// the index of one of the strings
in the slice, and then using that index
/// in the original array.
///
///     let streets = ["Adams",
"Bryant", "Channing", "Douglas",
"Evarts"]
///     let streetsSlice =
streets[2 ..< streets.endIndex]
///     print(streetsSlice)
///     // Prints "["Channing",
"Douglas", "Evarts"]"
///
///     let index =
streetsSlice.firstIndex(of:
"Evarts")    // 4
///     print(streets[index!])
///     // Prints "Evarts"
///
/// - Parameter bounds: A range of
the collection's indices. The bounds of
/// the range must be valid indices
of the collection.
///
/// - Complexity: O(1)
public subscript(bounds: Range<Int>)
-> SubviewsCollectionSlice { get }

/// The position of the first element
```

in a nonempty collection.

```
///
/// If the collection is empty,
`startIndex` is equal to `endIndex`.
public var startIndex: Int { get }

    /// The collection's "past the end"
position---that is, the position one
    /// greater than the last valid
subscript argument.
///
/// When you need a range that
includes the last element of a
collection, use
    /// the half-open range operator
(`..<`) with `endIndex`. The `..<`
operator
    /// creates a range that doesn't
include the upper bound, so it's always
    /// safe to use with `endIndex`. For
example:
///
///     let numbers = [10, 20, 30,
40, 50]
    ///     if let index =
numbers.firstIndex(of: 30) {
        ///         print(numbers[index ..<
numbers.endIndex])
    ///     }
    ///     // Prints "[30, 40, 50]"
///
    /// If the collection is empty,
`endIndex` is equal to `startIndex`.
```

```
public var endIndex: Int { get }

    /// A type representing the
    sequence's elements.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias Element = Subview

    /// A type that represents a position
    in the collection.
    ///
    /// Valid indices consist of the
    position of every element and a
    /// "past the end" position that's
    not valid for use as a subscript
    /// argument.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias Index = Int

    /// A type that represents the
    indices that are valid for subscripting
    the
    /// collection, in ascending order.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias Indices = Range<Int>

    /// A type that provides the
    collection's iteration interface and
```

```
    /// encapsulates its iteration state.  
    ///  
    /// By default, a collection conforms  
    to the `Sequence` protocol by  
    /// supplying `IndexingIterator` as  
    its associated `Iterator`  
    /// type.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias Iterator =  
IndexingIterator<SubviewsCollection>  
  
    /// A collection representing a  
    contiguous subrange of this collection's  
    /// elements. The subsequence shares  
    indices with the original collection.  
    ///  
    /// The default subsequence type for  
    collections that don't define their own  
    /// is `Slice`.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias SubSequence =  
SubviewsCollectionSlice  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension SubviewsCollection : View {  
  
    /// The type of view representing the
```

```
body of this view.  
    ///  
    /// When you create a custom view,  
Swift infers this type from your  
    /// implementation of the required  
``View/body-swift.property`` property.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias Body = Never  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
public struct SubviewsCollectionSlice :  
RandomAccessCollection {  
  
    /// Accesses the element at the  
specified position.  
    ///  
    /// The following example accesses an  
element of an array through its  
    /// subscript to print its value:  
    ///  
    ///     var streets = ["Adams",  
"Bryant", "Channing", "Douglas",  
"Evarts"]  
    ///     print(streets[1])  
    ///     // Prints "Bryant"  
    ///  
    /// You can subscript a collection  
with any valid index other than the  
    /// collection's end index. The end
```

```
index refers to the position one past
    /// the last element of a collection,
so it doesn't correspond with an
    /// element.
    ///
    /// - Parameter position: The
position of the element to access.
`position`
    /// must be a valid index of the
collection that is not equal to the
    /// `endIndex` property.
    ///
    /// - Complexity: O(1)
public subscript(index: Int) ->
Subview { get }

    /// Accesses a contiguous subrange of
the collection's elements.
    ///
    /// The accessed slice uses the same
indices for the same elements as the
    /// original collection uses. Always
use the slice's `startIndex` property
    /// instead of assuming that its
indices start at a particular value.
    ///
    /// This example demonstrates getting
a slice of an array of strings, finding
    /// the index of one of the strings
in the slice, and then using that index
    /// in the original array.
    ///
    ///     let streets = ["Adams",
```

```
"Bryant", "Channing", "Douglas",
"Everts"]
    ///      let streetsSlice =
streets[2 ..< streets.endIndex]
    ///      print(streetsSlice)
    ///      // Prints "["Channing",
"Bryant", "Channing", "Douglas", "Everts"]"
    ///
    ///      let index =
streetsSlice.firstIndex(of:
"Everts")      // 4
    ///      print(streets[index!])
    ///      // Prints "Everts"
    ///
    /// - Parameter bounds: A range of
the collection's indices. The bounds of
    ///      the range must be valid indices
of the collection.
    ///
    /// - Complexity: O(1)
public subscript(bounds: Range<Int>)
-> SubviewsCollectionSlice { get }

    /// The position of the first element
in a nonempty collection.
    ///
    /// If the collection is empty,
`startIndex` is equal to `endIndex`.
public var startIndex: Int { get }

    /// The collection's "past the end"
position---that is, the position one
    /// greater than the last valid
```

subscript argument.

```
///
/// When you need a range that
includes the last element of a
collection, use
    /// the half-open range operator
(`..<`) with `endIndex`. The `..<`
operator
    /// creates a range that doesn't
include the upper bound, so it's always
    /// safe to use with `endIndex`. For
example:
///
///     let numbers = [10, 20, 30,
40, 50]
    ///     if let index =
numbers.firstIndex(of: 30) {
        ///         print(numbers[index ..<
numbers.endIndex])
    ///     }
    ///     // Prints "[30, 40, 50]"
    ///
    /// If the collection is empty,
`endIndex` is equal to `startIndex`.
public var endIndex: Int { get }

    /// A type representing the
sequence's elements.
@available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
public typealias Element = Subview
```

```
    /// A type that represents a position
    in the collection.
    /**
     /// Valid indices consist of the
     position of every element and a
     /// "past the end" position that's
     not valid for use as a subscript
     /// argument.
     @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias Index = Int

    /// A type that represents the
    indices that are valid for subscripting
    the
    /// collection, in ascending order.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias Indices = Range<Int>

    /// A type that provides the
    collection's iteration interface and
    /// encapsulates its iteration state.
    /**
     /// By default, a collection conforms
     to the `Sequence` protocol by
     /// supplying `IndexingIterator` as
     its associated `Iterator`
     /// type.
     @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
```

```
*)  
    public typealias Iterator =  
IndexingIterator<SubviewsCollectionSlice>  
  
        /// A collection representing a  
contiguous subrange of this collection's  
        /// elements. The subsequence shares  
indices with the original collection.  
        ///  
        /// The default subsequence type for  
collections that don't define their own  
        /// is `Slice`.  
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
    public typealias SubSequence =  
SubviewsCollectionSlice  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension SubviewsCollectionSlice : View  
{  
  
    /// The type of view representing the  
body of this view.  
    ///  

```

```
    *)
        public typealias Body = Never
    }

/// A symbol rendering mode.
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
public struct SymbolRenderingMode : Sendable {

    /// A mode that renders symbols as a
    /// single layer filled with the
    /// foreground style.
    ///
    /// For example, you can render a
    /// filled exclamation mark triangle in
    /// purple:
    ///
    ///     Image(systemName:
    /// "exclamationmark.triangle.fill")
    ///         .symbolRenderingMode(.monochrom
    ///         .foregroundStyle(Color.purple)
    public static let monochrome: SymbolRenderingMode

    /// A mode that renders symbols as
    /// multiple layers with their inherit
    /// styles.
    ///
    /// The layers may be filled with
    /// their own inherent styles, or the
```

```
    /// foreground style. For example,  
you can render a filled exclamation  
    /// mark triangle in its inherent  
colors, with yellow for the triangle and  
    /// white for the exclamation mark:  
    ///  
    ///     Image(systemName:  
"exclamationmark.triangle.fill")  
    ///             .symbolRenderingMode(.mul  
ticolor)  
public static let multicolor:  
SymbolRenderingMode
```

```
    /// A mode that renders symbols as  
multiple layers, with different opacities  
    /// applied to the foreground style.  
    ///  
    /// SwiftUI fills the first layer  
with the foreground style, and the others  
    /// the secondary, and tertiary  
variants of the foreground style. You can  
    /// specify these styles explicitly  
using the ``View/foregroundStyle(_:_:)``  
    /// and  
``View/foregroundStyle(_:_:_:)``  
modifiers. If you only specify  
    /// a primary foreground style,  
SwiftUI automatically derives  
    /// the others from that style. For  
example, you can render a filled  
    /// exclamation mark triangle with  
purple as the tint color for the  
    /// exclamation mark, and lower
```

opacity purple for the triangle:

```
///
///      Image(systemName:
"exclamationmark.triangle.fill")
    ///
    .symbolRenderingMode(.hie
archical)
    ///
    .foregroundStyle(Color.pu
rple)
public static let hierarchical:
SymbolRenderingMode
```

/// A mode that renders symbols as multiple layers, with different styles

/// applied to the layers.

///

/// In this mode SwiftUI maps each successively defined layer in the image

/// to the next of the primary, secondary, and tertiary variants of the

/// foreground style. You can specify these styles explicitly using the

/// ``View/foregroundStyle(\_:\_:)``

and ``View/foregroundStyle(\_:\_:\_:)``

/// modifiers. If you only specify a primary foreground style, SwiftUI

/// automatically derives the others from that style. For example, you can

/// render a filled exclamation mark triangle with yellow as the tint color

/// for the exclamation mark, and fill the triangle with cyan:

///

/// Image(systemName:

```
"exclamationmark.triangle.fill")
    /**
     * symbolRenderingMode(.pal
     * ette)
    /**
     * foregroundStyle(Color.ye
     * llow, Color.cyan)
    /**
     * You can also omit the symbol
     * rendering mode, as specifying multiple
     * foreground styles implies
     * switching to palette rendering mode:
    /**
     * Image(systemName:
"exclamationmark.triangle.fill")
    /**
     * foregroundStyle(Color.ye
     * llow, Color.cyan)
public static let palette:
SymbolRenderingMode
}
```

```
/// A variant of a symbol.
///
/// Many of the
///
<doc://com.apple.documentation/design/hum
an-interface-guidelines/sf-symbols>
/// that you can add to your app using an
``Image`` or a ``Label`` instance
/// have common variants, like a filled
version or a version that's
/// contained within a circle. The
symbol's name indicates the variant:
///
/// VStack(alignment: .leading) {
```

```
///             Label("Default", systemImage:  
"heart")  
///             Label("Fill", systemImage:  
"heart.fill")  
///             Label("Circle", systemImage:  
"heart.circle")  
///             Label("Circle Fill",  
systemImage: "heart.circle.fill")  
///         }  
///  
/// ! [A screenshot showing an outlined  
heart, a filled heart, a heart in  
/// a circle, and a filled heart in a  
circle, each with a text label  
/// describing the symbol.]  
(SymbolVariants-1)  
///  
/// You can configure a part of your view  
hierarchy to use a particular variant  
/// for all symbols in that view and its  
child views using `SymbolVariants`.  
/// Add the ``View/symbolVariant(_:)``  
modifier to a view to set a variant  
/// for that view's environment. For  
example, you can use the modifier to  
/// create the same set of labels as in  
the example above, using only the  
/// base name of the symbol in the label  
declarations:  
///  
///     VStack(alignment: .leading) {  
///         Label("Default", systemImage:  
"heart")
```

```
///             Label("Fill", systemImage:  
"heart")  
///                     .symbolVariant(.fill)  
///             Label("Circle", systemImage:  
"heart")  
///                     .symbolVariant(.circle)  
///             Label("Circle Fill",  
systemImage: "heart")  
///                     .symbolVariant(.circle.fi  
ll)  
///     }  
///  
/// Alternatively, you can set the  
variant in the environment directly by  
/// passing the  
``EnvironmentValues/symbolVariants``  
environment value to the  
/// ``View/environment(_:_:)`` modifier:  
///  
///     Label("Fill", systemImage:  
"heart")  
///             .environment(\.symbolVariants  
, .fill)  
///  
/// SwiftUI sets a variant for you in  
some environments. For example, SwiftUI  
/// automatically applies the  
``SymbolVariants/fill-  
swift.type.property``  
/// symbol variant for items that appear  
in the `content` closure of the  
///  
``View/swipeActions(edge:allowsFullSwipe:
```

```
content:```
/// method, or as the tab bar items of a
``TabView``.
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
public struct SymbolVariants : Hashable,
Sendable {

    /// No variant for a symbol.
    ///
    /// Using this variant with the
    ``View/symbolVariant(_:)`` modifier
    doesn't
        /// have any effect. Instead, to show
    a symbol that ignores the current
        /// variant, directly set the
    ``EnvironmentValues/symbolVariants``
        /// environment value to `none` using
    the ``View/environment(_:_)``
        /// modifer:
    ///
    ///     HStack {
    ///         Image(systemName:
    "heart")
    ///         Image(systemName:
    "heart")
    ///             .environment(\.symbol
Variants, .none)
    ///         }
    ///         .symbolVariant(.fill)
    ///
    /// ! [A screenshot of two heart
symbols. The first is filled while the
```

```
    /// second is outlined.]  
(SymbolVariants-none-1)  
    public static let none:  
SymbolVariants  
  
    /// A variant that encapsulates the  
symbol in a circle.  
    ///  
    /// Use this variant with a call to  
the ``View/symbolVariant(_:)`` modifier  
    /// to draw symbols in a circle, for  
those symbols that have a circle  
    /// variant:  
    ///  
    ///     VStack(spacing: 20) {  
    ///         HStack(spacing: 20) {  
    ///             Image(systemName:  
"flag")  
    ///             Image(systemName:  
"heart")  
    ///             Image(systemName:  
"bolt")  
    ///             Image(systemName:  
"star")  
    ///         }  
    ///         HStack(spacing: 20) {  
    ///             Image(systemName:  
"flag")  
    ///             Image(systemName:  
"heart")  
    ///             Image(systemName:  
"bolt")  
    ///             Image(systemName:
```

```
    /// A variant that encapsulates the
    symbol in a square.
    ///
    /// Use this variant with a call to
    the ``View/symbolVariant(_:)`` modifier
    /// to draw symbols in a square, for
    those symbols that have a square
    /// variant:
    ///
    ///     VStack(spacing: 20) {
    ///         HStack(spacing: 20) {
    ///             Image(systemName:
    "flag")
    ///             Image(systemName:
    "heart")
    ///             Image(systemName:
    "bolt")
    ///             Image(systemName:
```

```
    /// A variant that encapsulates the
    symbol in a rectangle.
    ///
    /// Use this variant with a call to
    the ``View/symbolVariant(_:)`` modifier
    /// to draw symbols in a rectangle,
    for those symbols that have a rectangle
    /// variant:
```

```
///  
///      VStack(spacing: 20) {  
///          HStack(spacing: 20) {  
///              Image(systemName:  
"plus")  
///              Image(systemName:  
"minus")  
///              Image(systemName:  
"xmark")  
///              Image(systemName:  
"checkmark")  
///          }  
///          HStack(spacing: 20) {  
///              Image(systemName:  
"plus")  
///              Image(systemName:  
"minus")  
///              Image(systemName:  
"xmark")  
///              Image(systemName:  
"checkmark")  
///      }  
///      .symbolVariant(.rectangle  
)  
///  }  
///  ///  ///  ! [A screenshot showing two rows  
of four symbols each. Both rows contain  
/// a plus sign, a minus sign, a  
multiplication sign, and a check mark.  
/// The symbols in the second row are  
versions of the symbols in the first  
/// row, but each is enclosed in a
```

```
rectangle.](SymbolVariants-rectangle-1)
    public static let rectangle:
SymbolVariants

        /// A version of the variant that's
encapsulated in a circle.
        /**
         /// Use this property to modify a
variant like ``fill-swift.property``
         /// so that it's also contained in a
circle:
        /**
         ///     Label("Fill Circle",
systemImage: "bolt")
        /**
             .symbolVariant(.fill.circ
le)
        /**
         /// ! [A screenshot of a label that
shows a bolt in a filled circle
         /// beside the words Fill Circle.]
```

(SymbolVariants-circle-2)

```
    public var circle: SymbolVariants {
get }
```

  

```
        /// A version of the variant that's
encapsulated in a square.
        /**
         /// Use this property to modify a
variant like ``fill-swift.property``
         /// so that it's also contained in a
square:
        /**
         ///     Label("Fill Square",
```

```
systemImage: "star")
    /**
     * symbolVariant(.fill.square)
    /**
     * ! [A screenshot of a label that
     * shows a star in a filled square
     * beside the words Fill Square.]
```

(SymbolVariants-square-2)

```
public var square: SymbolVariants {
get }
```

/// A version of the variant that's encapsulated in a rectangle.

```
/**
 * Use this property to modify a
 * variant like ``fill-swift.property``
 * so that it's also contained in a
 * rectangle:
```

```
/**
 * Label("Fill Rectangle",
systemImage: "plus")
    /**
     * symbolVariant(.fill.rectangle)
    /**
     * ! [A screenshot of a label that
     * shows a plus sign in a filled rectangle
     * beside the words Fill Rectangle.]
```

(SymbolVariants-rectangle-2)

```
public var rectangle: SymbolVariants
{ get }
```

/// A variant that fills the symbol.

```
///
```

```
    /// Use this variant with a call to
the ``View/symbolVariant(_:)`` modifier
    /// to draw filled symbols, for those
symbols that have a filled variant:
    ///
    ///     VStack(spacing: 20) {
    ///         HStack(spacing: 20) {
    ///             Image(systemName:
"flag")
    ///             Image(systemName:
"heart")
    ///             Image(systemName:
"bolt")
    ///             Image(systemName:
"star")
    ///         }
    ///         HStack(spacing: 20) {
    ///             Image(systemName:
"flag")
    ///             Image(systemName:
"heart")
    ///             Image(systemName:
"bolt")
    ///             Image(systemName:
"star")
    ///         }
    ///         .symbolVariant(.fill)
    ///     }
    ///
    ///     ! [A screenshot showing two rows
of four symbols each. Both rows contain
    /// a flag, a heart, a bolt, and a
star. The symbols in the second row are
```

```
    /// filled version of the symbols in
    the first row.](SymbolVariants-fill-1)
    public static let fill:
SymbolVariants

    /// A filled version of the variant.
    ///
    /// Use this property to modify a
shape variant like
    /// ``circle-swift.type.property`` so
that it's also filled:
    ///
    ///     Label("Circle Fill",
systemImage: "flag")
    ///         .symbolVariant(.circle.fi
ll)
    ///
    /// ![A screenshot of a label that
shows a flag in a filled circle
    /// beside the words Circle Fill.]
```

(SymbolVariants-fill-2)

```
    public var fill: SymbolVariants { get
}
```

  

```
    /// A variant that draws a slash
through the symbol.
    ///
    /// Use this variant with a call to
the ``View/symbolVariant(_:)`` modifier
    /// to draw symbols with a slash, for
those symbols that have such a
    /// variant:
    ///
```

```
    ///      VStack(spacing: 20) {
    ///          HStack(spacing: 20) {
    ///              Image(systemName:
    "flag")
    ///              Image(systemName:
    "heart")
    ///              Image(systemName:
    "bolt")
    ///              Image(systemName:
    "star")
    ///          }
    ///          HStack(spacing: 20) {
    ///              Image(systemName:
    "flag")
    ///              Image(systemName:
    "heart")
    ///              Image(systemName:
    "bolt")
    ///              Image(systemName:
    "star")
    ///          }
    ///          .symbolVariant(.slash)
    ///      }
    ///      /**
    ///          ! [A screenshot showing two rows
    ///          of four symbols each. Both rows contain
    ///          a flag, a heart, a bolt, and a
    ///          star. A slash is superimposed over
    ///          all the symbols in the second
    ///          row.] (SymbolVariants-slash-1)
    public static let slash:
SymbolVariants
```

```
    /// A slashed version of the variant.  
    ///  
    /// Use this property to modify a  
shape variant like  
    /// ``circle-swift.type.property`` so  
that it's also covered by a slash:  
    ///  
    ///     Label("Circle Slash",  
systemImage: "flag")  
    ///             .symbolVariant(.circle.slash)  
    ///  
    /// ! [A screenshot of a label that  
shows a flag in a circle with a  
    /// slash over it beside the words  
Circle Slash.] (SymbolVariants-slash-2)  
    public var slash: SymbolVariants {  
get }  
  
    /// Returns a Boolean value that  
indicates whether the current variant  
    /// contains the specified variant.  
    ///  
    /// - Parameter other: A variant to  
look for in this variant.  
    /// - Returns: `true` if this variant  
contains `other`; otherwise,  
    /// `false`.  
    public func contains(_ other:  
SymbolVariants) -> Bool  
  
    /// 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:  
        SymbolVariants, b: SymbolVariants) ->  
        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 namespace for format styles that  
    implement designs used across Apple's  
    /// platforms.  
    @available(iOS 18.0, macOS 15.0, tvOS  
    18.0, watchOS 11.0, visionOS 2.0, *)  
    public enum SystemFormatStyle : Sendable  
    {
```

```
}
```

```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension SystemFormatStyle {  
  
    /// A system format style to refer to  
    // a date in the most natural way.  
    ///  
    /// The reference format is designed  
    // for referring to a  
    /// certain date in the most natural  
    // way possible. The concrete  
    /// format depends on the distance to  
    // the date. E.g. if an event  
    /// is one minute away, most people  
    // would refer to it in a  
    /// relative way, e.g. the event  
    // starts "in one minute". However,  
    /// if dates are really far away, it  
    becomes easier to refer to  
    /// them in an absolute way, e.g. the  
    // original iPhone was announced  
    /// in "January 2007".  
    ///  
    /// Use the style by initializing it  
    // with the date it should refer to and  
    /// format it with the point of  
    // reference, usually the current time.  
    public struct DateReference :  
Sendable {  
  
    /// Create a format style to
```

refer to a date in the most natural way.

///

/// - Parameters:

/// - date: The date this  
format references.

/// - allowedFields: The units  
of time that may be used in the format  
/// to express the reference.

The `thresholdField` is always assumed

/// to be allowed.

/// - maxFieldCount: The number  
of fields that can be shown at once

/// when using an absolute  
format. For example, January 9, 2007 is  
/// shown as `January 2007` by  
default, but as `January 9, 2007` if

/// the `maxFieldCount` is set  
to three. The style automatically

/// excludes more significant  
fields if their value is equal to the

/// value in the reference date  
and they are not necessary for the

/// format pattern, making room  
for less significant fields.

/// - thresholdField: The least  
precise field preserving which

/// warrants increasing the  
field count from one, i.e. switching from

/// the relative to the  
absolute representation.

**public init**(**to** date: Date,  
allowedFields:

Set<Date.RelativeFormatStyle.Field> =

```
[.year, .month, .day, .hour, .minute],  
maxFieldCount: Int = 2, thresholdField:  
Date.RelativeFormatStyle.Field = .day)  
  
        public func calendar(_ calendar:  
Calendar) ->  
SystemFormatStyle.DateReference  
    }  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension SystemFormatStyle {  
  
    /// A system format to display the  
    offset to a certain date.  
    public struct DateOffset : Sendable {  
  
        /// Create a format style to  
        display the offset to a certain date.  
        ///  
        /// The time format (`3:46`) is  
        used as long as only minutes and  
        /// seconds, or hours, minutes,  
        and second may be shown.  
        /// Otherwise, calendar units are  
        used, resulting in outputs like  
        /// `3 months, 11 days`.  
        ///  
        /// The offset to the `anchor` is  
        "positive" if the formatted  
        /// `date` is greater than the  
        given `anchor` specified here.
```

```
    /// Conversely, "negative"  
offsets are displayed if the input  
    /// `date` is smaller than the  
`anchor`.  
    ///  
    /// - Parameters:  
    ///   - anchor: The date the  
offset is measured to from the format  
    ///   input.  
    ///   - allowedFields: The units  
of time that may be used in the format  
    ///   to express the offset.  
    ///   - maxFieldCount: The number  
of fields that can be shown at once.  
    ///   For example, 1 hour, 34  
minutes, and 23 seconds is shown as  
    ///   `1 hour, 34 minutes` by  
default, but as `1 hour` if the  
    ///   `maxFieldCount` is set to  
one.  
    ///   - sign: The strategy for  
displaying a sign to signal whether the  
    ///   offset points toward the  
future or past.  
        public init(to anchor: Date,  
allowedFields:  
Set<Date.ComponentsFormatStyle.Field> =  
[.year, .month, .week, .day, .hour, .minu  
te, .second], maxFieldCount: Int = 2,  
sign:  
NumberFormatStyleConfiguration.SignDispla  
yStrategy = .automatic)
```

```
        public func calendar(_ calendar:  
Calendar) -> SystemFormatStyle.DateOffset  
    }  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension SystemFormatStyle {  
  
    /// The system timer format style.  
    public struct Timer : Sendable {  
  
        /// The type of data to format.  
        public typealias FormatInput =  
Date  
  
        /// Create a timer format style  
        // that counts down from the given interval.  
        ///  
        /// A timer styled display that  
        // counts from the given `timerInterval`  
        // down to zero.  
        ///  
        /// - Parameters:  
        /// - interval: The interval  
        // during which the timer counts down.  
        /// - showsHours: If true, the  
        // timer shows the hours as a separate  
        // element on the  
        /// formatted string, as long as  
        // the duration is at least one hour. If  
        // false, the  
        /// timer displays minute values
```

greater than sixty.

    /// - maxFieldCount: The number of fields that can be shown at once. For example,

        /// 1 hour, 34 minutes is shown as `1:34:00` by default, but as `1:34` if the

        /// `maxFieldCount` is set to two. The style automatically excludes more significant

        /// fields if their value is zero and they are not necessary for the format pattern, making

        /// room for less significant fields.

    /// - maxPrecision: The precision at which the input is formatted. E.g. by

        /// default, seconds are shown, making the maximum precision one second. Setting

        /// the maximum precision to `seconds(60)` would only allow hours and minutes to

        /// be shown.

```
public init(countingDownIn
interval: Range<Date>, showsHours: Bool =
true, maxFieldCount: Int = 3,
maxPrecision: Duration = .seconds(1))
```

    /// Create a timer format style that counts up to the given interval.

    ///

    /// A timer styled display that

counts from zero up to the given `timerInterval`.

    ///

    /// - Parameters:

        /// - interval: The interval during which the timer counts up.

        /// - showsHours: If true, the timer shows the hours as a separate element on the

            /// formatted string, as long as the duration is at least one hour. If false, the

            /// timer displays minute values greater than sixty.

        /// - maxFieldCount: The number of fields that can be shown at once. For example,

            /// 1 hour, 34 minutes is shown as `1:34:00` by default, but as `1:34` if the

            /// `maxFieldCount` is set to two. The style automatically excludes more significant

            /// fields if their value is zero and they are not necessary for the format pattern, making

            /// room for less significant fields.

        /// - maxPrecision: The precision at which the input is formatted. E.g. by

            /// default, seconds are shown, making the maximum precision one second. Setting

```
        /// the maximum precision to
`seconds(60)` would only allow hours
and minutes to
        /// be shown.
    public init(countingUpIn
interval: Range<Date>, showsHours: Bool =
true, maxFieldCount: Int = 3,
maxPrecision: Duration = .seconds(1))
    }
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension SystemFormatStyle {

    /// The system stopwatch format
style.
    public struct Stopwatch : Sendable {

        /// The type of data to format.
        public typealias FormatInput =
Date

        /// Create a stopwatch format
style.
        ///
        /// A stopwatch styled display
that starts counting up from zero at the
given
        /// `startDate`.
        ///
        /// - Parameters:
        /// - startDate: The date at
```

which the stopwatch starts counting.

    /// – showsHours: If true, the stopwatch shows the hours as a separate element on

        /// the formatted string, once the duration is at least one hour. If false, the

        /// stopwatch displays minute values greater than sixty.

    /// – maxFieldCount: The number of fields that can be shown at once. For example,

        /// 1 hour, 34 minutes is shown as `1:34:00` by default, but as `1:34` if the

        /// `maxFieldCount` is set to two. The style automatically excludes more significant

        /// fields if their value is zero and they are not necessary for the format pattern, making

        /// room for less significant fields.

    /// – maxPrecision: The precision at which the input is formatted. E.g. by

        /// default, two fractional digits are shown, making the maximum precision ten

        /// milliseconds. Setting the maximum precision to ` `.seconds(60)` would only allow

        /// hours and minutes to be shown.

```
        public init(startingAt startDate:  
Date, showsHours: Bool = true,  
maxFieldCount: Int = 4, maxPrecision:  
Duration = .milliseconds(10))  
    }  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension SystemFormatStyle.DateReference  
: FormatStyle {  
  
    /// Creates a `FormatOutput` instance  
    // from `value`.  
    public func format(_ referenceDate:  
Date) -> AttributedString  
  
    /// If the format allows selecting a  
    // locale, returns a copy of this format  
    // with the new locale set. Default  
    implementation returns an unmodified  
    self.  
    public func locale(_ locale: Locale)  
-> SystemFormatStyle.DateReference  
  
    /// 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:  

SystemFormatStyle.DateReference, b:
```

```
SystemFormatStyle.DateReference) -> Bool

    /// The type of data to format.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias FormatInput = Date

    /// The type of the formatted data.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias FormatOutput =
AttributedString

    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
    public func encode(to encoder: any
Encoder) throws

    /// 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 }
```

```
    /// Creates a new instance by  
decoding from the given decoder.
```

```
    ///
```

```
    /// This initializer throws an error  
if reading from the decoder fails, or
```

```
    /// if the data read is corrupted or  
otherwise invalid.
```

```
    ///
```

```
    /// - Parameter decoder: The decoder  
to read data from.
```

```
public init(from decoder: any  
Decoder) throws  
{
```

```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension SystemFormatStyle.DateReference  
: DiscreteFormatStyle {
```

```
    /// The next discretization boundary  
before the given input.  
    ///  
    /// Use this function to determine  
the next "smaller" input that warrants  
updating the formatted output.  
    /// The following example prints all  
possible outputs the format style can  
produce downwards starting  
    /// from the `startInput`:  
    ///  
    /// ````swift  
    /// var previousInput = startInput  
    /// while let nextInput =  
style.discreteInput(before:  
previousInput) {  
    ///  
print(style.format(nextInput))  
    ///     previousInput = nextInput  
    /// }  
    /// ````  
    ///  
    /// - Returns: For most `input`s, the  
method returns the "greatest" value  
"smaller" than  
    /// `input` for which the style  
produces a different  
``FormatStyle/FormatOutput``, or `nil`  
    /// if no such value exists. For some  
input values, the function may also  
return a value "smaller" than  
    /// `input` for which the style still
```

```
produces the same
``FormatStyle/FormatOutput`` as for
    /// `input`.
    public func discreteInput(before
referenceDate: Date) -> Date?

    /// The next discretization boundary
after the given input.
    ///
    /// Use this function to determine
the next "greater" input that warrants
updating the formatted output.
    /// The following example prints all
possible outputs the format style can
produce upwards starting
    /// from the `startInput`:
    ///
    /// ````swift
    /// var previousInput = startInput
    /// while let nextInput =
style.discreteInput(after: previousInput)
{
    ///
print(style.format(nextInput))
    ///     previousInput = nextInput
    /// }
    ///
    ///
    ///
    /// - Returns: For most `input`s, the
method returns the "smallest" value
"greater" than
    /// `input` for which the style
produces a different
```

```
``FormatStyle/FormatOutput``, or `nil`  
    /// if no such value exists. For some  
input values, the function may also  
return a value "greater" than  
    /// `input` for which the style still  
produces the same  
``FormatStyle/FormatOutput`` as for  
    /// `input`.  
public func discreteInput(after  
referenceDate: Date) -> Date?  
  
    /// The next input value before the  
given input.  
    ///  
    /// Use this function to determine if  
the return value provided by  
``discreteInput(after:)`` is  
    /// precise enough for your use case  
for any input `y`:  
    ///  
    /// ````swift  
    /// guard let x =  
style.discreteInput(after: y) else {  
    ///     return  
    /// }  
    ///  
    /// let z = style.input(before: x) ??  
y  
    /// ````  
    ///  
    /// If the distance between `z` and  
`x` is too large for the precision you  
require, you may want
```

```
    /// to manually probe
``FormatStyle/format(_:)`` at a higher
rate in that interval, as there is
    /// no guarantee for what the output
will be in that interval.

    ///
    /// - Returns: The next "smalller"
input value that can be represented by
    /// ``FormatStyle/FormatInput`` or an
underlying representation the format
style uses
    /// internally.
public func input(before
referenceDate: Date) -> Date?

    /// The next input value after the
given input.
    ///
    /// Use this function to determine if
the return value provided by
``discreteInput(before:)`` is
    /// precise enough for your use case
for any input `y`:
    ///
    /// ````swift
    /// guard let x =
style.discreteInput(before: y) else {
    ///     return
    /// }
    ///
    /// let z = style.input(after: x) ??
y
    /// ````
```

```
    /**
     * If the distance between `x` and
     * `z` is too large for the precision you
     * require, you may want
     *   /// to manually probe
     * ``FormatStyle/format(_:)`` at a higher
     * rate in that interval, as there is
     *   /// no guarantee for what the output
     * will be in that interval.
     *
     * - Returns: The next "greater"
     * input value that can be represented by
     *   /// ``FormatStyle/FormatInput`` or an
     * underlying representation the format
     * style uses
     *   /// internally.
    public func input(after
referenceDate: Date) -> Date?
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension SystemFormatStyle.DateOffset : FormatStyle {

    /// Creates a `FormatOutput` instance
    from `value`.
    public func format(_ referenceDate:
Date) -> AttributedString

    /// If the format allows selecting a
    locale, returns a copy of this format
    with the new locale set. Default
```

implementation returns an unmodified self.

```
public func locale(_ locale: Locale)  
-> SystemFormatStyle.DateOffset
```

```
    /// 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:
SystemFormatStyle.DateOffset, b:
SystemFormatStyle.DateOffset) -> Bool

    /**
     * The type of data to format.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias FormatInput = Date

    /**
     * The type of the formatted data.
    @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias FormatOutput =
AttributedString

    /**
     * Encodes this value into the given
     * encoder.
    /**
     * If the value fails to encode
     * anything, `encoder` will encode an empty
```

```
    /// keyed container in its place.  
    ///  
    /// This function throws an error if  
any values are invalid for the given  
    /// encoder's format.  
    ///  
    /// - Parameter encoder: The encoder  
to write data to.  
    public func encode(to encoder: any  
Encoder) throws  
  
    /// 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 }  
  
    /// Creates a new instance by  
decoding from the given decoder.  
    ///  
    /// This initializer throws an error  
if reading from the decoder fails, or
```

```
    /// if the data read is corrupted or
    otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
    to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension SystemFormatStyle.DateOffset :  
DiscreteFormatStyle {  
  
    /// The next discretization boundary
    before the given input.
    ///
    /// Use this function to determine
    the next "smaller" input that warrants
    updating the formatted output.
    /// The following example prints all
    possible outputs the format style can
    produce downwards starting
    /// from the `startInput`:
    ///
    /// ````swift
    /// var previousInput = startInput
    /// while let nextInput =
    style.discreteInput(before:
    previousInput) {
        ///
        print(style.format(nextInput))
        ///     previousInput = nextInput
```

```
    /**
     * Returns: For most `input`s, the
     * method returns the "greatest" value
     * "smaller" than
     *   /// `input` for which the style
     * produces a different
     *   ``FormatStyle/FormatOutput``, or `nil`
     *   /// if no such value exists. For some
     * input values, the function may also
     * return a value "smaller" than
     *   /// `input` for which the style still
     * produces the same
     *   ``FormatStyle/FormatOutput`` as for
     *   /// `input`.
public func discreteInput(before
referenceDate: Date) -> Date?

    /// The next discretization boundary
after the given input.
    /**
     /// Use this function to determine
the next "greater" input that warrants
updating the formatted output.
    /// The following example prints all
possible outputs the format style can
produce upwards starting
    /// from the `startInput`:
    /**
     /// ``swift
    /// var previousInput = startInput
    /// while let nextInput =
```

```
style.discreteInput(after: previousInput)
{
    /**
     print(style.format(nextInput))
     /**
      * previousInput = nextInput
      */
      ...
      /**
       /**
        * - Returns: For most `input`s, the
        method returns the "smallest" value
        "greater" than
        /**
         * `input` for which the style
        produces a different
        ``FormatStyle/FormatOutput``, or `nil`
        /**
         if no such value exists. For some
        input values, the function may also
        return a value "greater" than
        /**
         * `input` for which the style still
        produces the same
        ``FormatStyle/FormatOutput`` as for
        /**
         * `input`.
    public func discreteInput(after
referenceDate: Date) -> Date?

    /**
     The next input value before the
given input.
    /**
    /**
     Use this function to determine if
the return value provided by
``discreteInput(after:)`` is
    /**
     precise enough for your use case
for any input `y`:
    /**

```

```
    /// ````swift
    /// guard let x =
style.discreteInput(after: y) else {
    ///     return
    /// }
    ///
    /// let z = style.input(before: x) ??
y
    ///
    ///
    /// If the distance between `z` and
`x` is too large for the precision you
require, you may want
    /// to manually probe
``FormatStyle/format(_:)`` at a higher
rate in that interval, as there is
    /// no guarantee for what the output
will be in that interval.
    ///
    /// - Returns: The next "smalller"
input value that can be represented by
    /// ``FormatStyle/FormatInput`` or an
underlying representation the format
style uses
    /// internally.
public func input(before
referenceDate: Date) -> Date?

    /// The next input value after the
given input.
    ///
    /// Use this function to determine if
the return value provided by
```



```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension SystemFormatStyle.Timer :  
FormatStyle {  
  
    /// Creates a `FormatOutput` instance  
    // from `value`.  
    public func format(_ input: Date) ->  
AttributedString  
  
    /// If the format allows selecting a  
    // locale, returns a copy of this format  
    // with the new locale set. Default  
    implementation returns an unmodified  
    self.  
    public func locale(_ locale: Locale)  
-> SystemFormatStyle.Timer  
  
    /// 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:  
SystemFormatStyle.Timer, b:  
SystemFormatStyle.Timer) -> Bool  
  
    /// The type of the formatted data.  
@available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)  
public typealias FormatOutput =
```

## AttributedString

```
    /// Encodes this value into the given
encoder.
    ///
    /// If the value fails to encode
anything, `encoder` will encode an empty
    /// keyed container in its place.
    ///
    /// This function throws an error if
any values are invalid for the given
    /// encoder's format.
    ///
    /// - Parameter encoder: The encoder
to write data to.
public func encode(to encoder: any
Encoder) throws

    /// 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 hashCode: Int { get }

    /// Creates a new instance by
    decoding from the given decoder.
    ///
    /// This initializer throws an error
    if reading from the decoder fails, or
    /// if the data read is corrupted or
    otherwise invalid.
    ///
    /// - Parameter decoder: The decoder
    to read data from.
    public init(from decoder: any
Decoder) throws
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension SystemFormatStyle.Timer :
DiscreteFormatStyle {

    /// The next discretization boundary
    before the given input.
    ///
    /// Use this function to determine
    the next "smaller" input that warrants
    updating the formatted output.
    /// The following example prints all
    possible outputs the format style can
    produce downwards starting
    /// from the `startInput`:
    ///
    /// ````swift
```

```
    /// var previousInput = startInput
    /// while let nextInput =
style.discreteInput(before:
previousInput) {
    ///
print(style.format(nextInput))
    ///     previousInput = nextInput
    ///
    ///
    ///
    ///
    /// - Returns: For most `input`s, the
method returns the "greatest" value
"smaller" than
    /// `input` for which the style
produces a different
``FormatStyle/FormatOutput``, or `nil`
    // if no such value exists. For some
input values, the function may also
return a value "smaller" than
    /// `input` for which the style still
produces the same
``FormatStyle/FormatOutput`` as for
    /// `input`.
public func discreteInput(before
input: Date) -> Date?

    /// The next discretization boundary
after the given input.
    ///
    /// Use this function to determine
the next "greater" input that warrants
updating the formatted output.
    /// The following example prints all
```

possible outputs the format style can produce upwards starting

```
    /// from the `startInput`:  
    ///  
    /// ````swift  
    /// var previousInput = startInput  
    /// while let nextInput =  
style.discreteInput(after: previousInput)  
{  
    ///  
    print(style.format(nextInput))  
    ///     previousInput = nextInput  
    /// }  
    /// ``  
    ///  
    /// - Returns: For most `input`s, the method returns the "smallest" value "greater" than  
    /// `input` for which the style produces a different ``FormatStyle/FormatOutput``, or `nil`  
    /// if no such value exists. For some input values, the function may also return a value "greater" than  
    /// `input` for which the style still produces the same ``FormatStyle/FormatOutput`` as for  
    /// `input`.  
    public func discreteInput(after input: Date) -> Date?  
}
```

**@available(iOS 18.0, macOS 15.0, tvOS**

```
18.0, watchOS 11.0, visionOS 2.0, *)
extension SystemFormatStyle.Stopwatch : FormatStyle {

    /// Creates a `FormatOutput` instance from `value`.
    public func format(_ input: Date) -> AttributedString

    /// If the format allows selecting a locale, returns a copy of this format with the new locale set. Default implementation returns an unmodified self.
    public func locale(_ locale: Locale) -> SystemFormatStyle.Stopwatch

    /// 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:  
SystemFormatStyle.Stopwatch, b:  
SystemFormatStyle.Stopwatch) -> Bool
```

```
    /// The type of the formatted data.
```

```
    @available(iOS 18.0, tvOS 18.0,  
watchOS 11.0, visionOS 2.0, macOS 15.0,  
*)
```

```
    public typealias FormatOutput =  
AttributedString
```

```
    /// 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 }  
}
```

```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension SystemFormatStyle.Stopwatch :  
DiscreteFormatStyle {
```

```
    /// The next discretization boundary  
    before the given input.
```

```
    ///  
    /// Use this function to determine  
    the next "smaller" input that warrants  
    updating the formatted output.
```

```
    /// The following example prints all  
    possible outputs the format style can  
    produce downwards starting
```

```
    /// from the `startInput`:  
    ///  
    /// ````swift  
    /// var previousInput = startInput  
    /// while let nextInput =  
style.discreteInput(before:  
previousInput) {  
    ///  
    print(style.format(nextInput))  
    ///     previousInput = nextInput  
    /// }  
    /// ````  
    ///  
    /// - Returns: For most `input`s, the  
method returns the "greatest" value  
"smaller" than  
    /// `input` for which the style  
produces a different  
``FormatStyle/FormatOutput``, or `nil`  
    /// if no such value exists. For some  
input values, the function may also  
return a value "smaller" than  
    /// `input` for which the style still  
produces the same  
``FormatStyle/FormatOutput`` as for  
    /// `input`.  
public func discreteInput(before  
input: Date) -> Date?  
  
    /// The next discretization boundary  
after the given input.  
    ///  
    /// Use this function to determine
```

the next "greater" input that warrants updating the formatted output.

    /// The following example prints all possible outputs the format style can produce upwards starting

```
    /// from the `startInput`:  
    ///  
    /// ````swift  
    /// var previousInput = startInput  
    /// while let nextInput =  
style.discreteInput(after: previousInput)  
{  
    ///  
print(style.format(nextInput))  
    ///     previousInput = nextInput  
    /// }..  
    /// ``..  
    ///  
    /// - Returns: For most `input`s, the method returns the "smallest" value "greater" than  
    /// `input` for which the style produces a different ``FormatStyle/FormatOutput``, or `nil`  
    /// if no such value exists. For some input values, the function may also return a value "greater" than  
    /// `input` for which the style still produces the same ``FormatStyle/FormatOutput`` as for  
    /// `input`.  
    public func discreteInput(after  
input: Date) -> Date?
```

```
}
```

```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension SystemFormatStyle.Stopwatch {  
  
    /// Creates a new instance by  
    decoding from the given decoder.  
    ///  
    /// This initializer throws an error  
    if reading from the decoder fails, or  
    /// if the data read is corrupted or  
    otherwise invalid.  
    ///  
    /// - Parameter decoder: The decoder  
    to read data from.  
    public init(from decoder: any  
Decoder) throws  
  
    /// Encodes this value into the given  
    encoder.  
    ///  
    /// If the value fails to encode  
    anything, `encoder` will encode an empty  
    /// keyed container in its place.  
    ///  
    /// This function throws an error if  
    any values are invalid for the given  
    /// encoder's format.  
    ///  
    /// - Parameter encoder: The encoder  
    to write data to.  
    public func encode(to encoder: any
```

```
Encoder) throws
```

```
}
```

```
/// A gesture that recognizes one or more  
taps.
```

```
///
```

```
/// To recognize a tap gesture on a view,  
create and configure the gesture, and  
/// then add it to the view using the  
``View/gesture(_:_including:)`` modifier.
```

```
/// The following code adds a tap gesture  
to a ``Circle`` that toggles the color  
/// of the circle:
```

```
///
```

```
///     struct TapGestureView: View {  
///         @State private var tapped =  
false
```

```
///
```

```
///         var tap: some Gesture {  
///             TapGesture(count: 1)  
///                 .onEnded { _ in  
self.tapped = !self.tapped }  
///         }
```

```
///
```

```
///         var body: some View {  
///             Circle()  
///                 .fill(self.tapped ?  
Color.blue : Color.red)  
///                     .frame(width: 100,  
height: 100, alignment: .center)  
///                         .gesture(tap)  
///         }  
///     }
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
16.0, watchOS 6.0, *)  
public struct TapGesture : Gesture {  
  
    /// The required number of tap  
    events.  
    public var count: Int  
  
    /// Creates a tap gesture with the  
    number of required taps.  
    /// - Parameter count: The required  
    number of taps to complete the tap  
    /// gesture.  
    public init(count: Int = 1)  
  
    /// The type of gesture representing  
    the body of `Self`.  
    @available(iOS 13.0, tvOS 16.0,  
    watchOS 6.0, macOS 10.15, *)  
    public typealias Body = Never  
  
    /// The type representing the  
    gesture's value.  
    @available(iOS 13.0, tvOS 16.0,  
    watchOS 6.0, macOS 10.15, *)  
    public typealias Value = Void  
}  
  
/// A view that displays one or more  
lines of read-only text.  
///  
/// A text view draws a string in your
```

app's user interface using a  
/// ``Font/body`` font that's appropriate  
for the current platform. You can  
/// choose a different standard font,  
like ``Font/title`` or ``Font/caption``,  
/// using the ``View/font(\_:)`` view  
modifier.

```
///  
///     Text("Hamlet")  
///         .font(.title)  
  
/// ! [A text view showing the name  
"Hamlet" in a title  
/// font.] (SwiftUI-Text-title.png)  
  
/// If you need finer control over the  
styling of the text, you can use the same  
/// modifier to configure a system font  
or choose a custom font. You can also  
/// apply view modifiers like  
``Text/bold()`` or ``Text/italic()`` to  
further  
/// adjust the formatting.  
  
///     Text("by William Shakespeare")  
///         .font(.system(size: 12,  
weight: .light, design: .serif))  
///             .italic()  
  
/// ! [A text view showing by William  
Shakespeare in a 12 point, light, italic,  
/// serif font.] (SwiftUI-Text-font.png)  
///
```

```
/// To apply styling within specific
/// portions of the text, you can create
/// the text view from an
///
<doc://com.apple.documentation/documentation/Foundation/AttributedString>,
/// which in turn allows you to use
Markdown to style runs of text. You can
/// mix string attributes and SwiftUI
modifiers, with the string attributes
/// taking priority.
///
///     let attributedString = try!
AttributedString(
///         markdown: "_Hamlet_ by
William Shakespeare")
///
///     var body: some View {
///         Text(attributedString)
///             .font(.system(size: 12,
weight: .light, design: .serif))
///     }
///
/// (![A text view showing Hamlet by
William Shakespeare in a 12 point, light,
/// serif font, with the title Hamlet in
italics.](SwiftUI-Text-attributed.png)
///
/// A text view always uses exactly the
amount of space it needs to display its
/// rendered contents, but you can affect
the view's layout. For example, you
/// can use the
```

```
``View/frame(width:height:alignment:)``
modifier to propose
/// specific dimensions to the view. If
the view accepts the proposal but the
/// text doesn't fit into the available
space, the view uses a combination of
/// wrapping, tightening, scaling, and
truncation to make it fit. With a width
/// of `100` points but no constraint on
the height, a text view might wrap a
/// long string:
///
///     Text("To be, or not to be, that
is the question:")
///         .frame(width: 100)
///
/// ! [A text view showing a quote from
Hamlet split over three
/// lines.] (SwiftUI-Text-split.png)
///
/// Use modifiers like
``View/lineLimit(_:)``,
``View/allowsTightening(_:)``,
/// ``View/minimumScaleFactor(_:)``, and
``View/truncationMode(_:)`` to
/// configure how the view handles space
constraints. For example, combining a
/// fixed width and a line limit of `1`
results in truncation for text that
/// doesn't fit in that space:
///
///     Text("Brevity is the soul of
wit.")
```

```
///         .frame(width: 100)
///         .lineLimit(1)
///
/// ! [A text view showing a truncated
quote from Hamlet starting Brevity is t
/// and ending with three dots.] (SwiftUI-
Text-truncated.png)
///
/// ### Localizing strings
///
/// If you initialize a text view with a
string literal, the view uses the
///
``Text/init(_:tableName:bundle:comment:)``
` initializer, which interprets the
/// string as a localization key and
searches for the key in the table you
/// specify, or in the default table if
you don't specify one.
///
///     Text("pencil") // Searches the
default table in the main bundle.
///
/// For an app localized in both English
and Spanish, the above view displays
/// "pencil" and "lápis" for English and
Spanish users, respectively. If the
/// view can't perform localization, it
displays the key instead. For example,
/// if the same app lacks Danish
localization, the view displays "pencil"
for
/// users in that locale. Similarly, an
```

```
app that lacks any localization
/// information displays "pencil" in any
locale.
///
/// To explicitly bypass localization for
a string literal, use the
/// ``Text/init(verbatim:)`` initializer.
///
///     Text(verbatim: "pencil") //
Displays the string "pencil" in any
locale.
///
/// If you initialize a text view with a
variable value, the view uses the
/// ``Text/init(_:)--9d1g4`` initializer,
which doesn't localize the string.
However,
/// you can request localization by
creating a ``LocalizedStringKey``
instance
/// first, which triggers the
``Text/init(_:tableName:bundle:comment:)``
`

/// initializer instead:
///
///     // Don't localize a string
variable...
///     Text(writingImplement)
///
///     // ...unless you explicitly
convert it to a localized string key.
///
Text(LocalizedStringKey(writingImplement))
```

```
)  
///  
/// When localizing a string variable,  
you can use the default table by omitting  
/// the optional initialization  
parameters – as in the above example –  
just like  
/// you might for a string literal.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct Text : Equatable,  
Sendable {  
  
    /// Creates a text view that displays  
a string literal without localization.  
    ///  
    /// Use this initializer to create a  
text view with a string literal without  
    /// performing localization:  
    ///  
    ///     Text(verbatim: "pencil") //  
Displays the string "pencil" in any  
locale.  
    ///  
    /// If you want to localize a string  
literal before displaying it, use the  
    ///  
``Text/init(_:tableName:bundle:comment:)``  
` initializer instead. If you  
    /// want to display a string  
variable, use the ``Text/init(_:)``-9d1g4``  
    /// initializer, which also bypasses  
localization.
```

```
///  
/// - Parameter content: A string to  
display without localization.  
@inlinable public init(verbatim  
content: String)  
  
    /// Creates a text view that displays  
a stored string without localization.  
    ///  
    /// Use this initializer to create a  
text view that displays – without  
    /// localization – the text in a  
string variable.  
    ///  
    ///     Text(someString) // Displays  
the contents of `someString` without  
localization.  
    ///  
    /// SwiftUI doesn't call the  
`init(_:)` method when you initialize a  
text  
    /// view with a string literal as the  
input. Instead, a string literal  
    /// triggers the  
``Text/init(_:tableName:bundle:comment:)``  
` method – which  
    /// treats the input as a  
``LocalizedStringKey`` instance – and  
attempts to  
    /// perform localization.  
    ///  
    /// By default, SwiftUI assumes that  
you don't want to localize stored
```

```
    /// strings, but if you do, you can
    first create a localized string key from
    /// the value, and initialize the
    text view with that. Using a key as input
    /// triggers the
``Text/init(_:tableName:bundle:comment:)``
` method instead.
    ///
    /// - Parameter content: The string
    value to display without localization.
    public init<S>(_ content: S) where
S : StringProtocol

    /// 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: Text, b:
Text) -> Bool
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Text {
```

```
    /// Controls the way text size  
variants are chosen.  
    ///  
    /// Certain types of text, such as  
``Text(_:format:)``, can generate strings  
of  
    /// different size to better fit the  
available space. By default, all text  
uses the  
    /// widest available variant. Setting  
the variant to be  
    ///  
``TextVariantPreference/sizeDependent``  
allows the text to take the available  
    /// space into account when choosing  
what content to display.  
    @available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
    public func textVariant<V>(  
        preference: V) -> some View where V :  
TextVariantPreference  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Text {  
  
    /// Creates a text view that displays  
localized content identified by a key.  
    ///  
    /// Use this initializer to look for  
the `key` parameter in a localization
```

```
    /// table and display the associated
    string value in the initialized text
    /// view. If the initializer can't
    find the key in the table, or if no table
    /// exists, the text view displays
    the string representation of the key
    /// instead.
    ///
    ///     Text("pencil") // Localizes
    the key if possible, or displays "pencil"
    if not.
    ///
    /// When you initialize a text view
    with a string literal, the view triggers
    /// this initializer because it
    assumes you want the string localized,
    even
    /// when you don't explicitly specify
    a table, as in the above example. If
    /// you haven't provided localization
    for a particular string, you still get
    /// reasonable behavior, because the
    initializer displays the key, which
    /// typically contains the
    unlocalized string.
    ///
    /// If you initialize a text view
    with a string variable rather than a
    /// string literal, the view triggers
    the ``Text/init(_):-9d1g4``
    /// initializer instead, because it
    assumes that you don't want localization
    /// in that case. If you do want to
```

```
localize the value stored in a string
    /// variable, you can choose to call
the `init(_:tableName:bundle:comment:)`  

    /// initializer by first creating a
``LocalizedStringKey`` instance from the
    /// string variable:  

    ///  

    ///  

    ///  

Text(LocalizedStringKey(someString)) //  

Localizes the contents of `someString`.
    ///  

    /// If you have a string literal that
you don't want to localize, use the
    /// ``Text/init(verbatim:)``  

initializer instead.  

    ///  

    /// ### Styling localized strings  

with markdown
    ///  

    /// If the localized string or the
fallback key contains Markdown, the
    /// view displays the text with
appropriate styling. For example,
consider
    /// an app with the following entry
in its Spanish localization file:
    ///  

    ///     "_Please visit our [website]  

(https://www.example.com)._" = "_Visita  

nuestro [sitio web]  

(https://www.example.com)._";  

    ///  

    /// You can create a `Text` view with
```

the Markdown-formatted base language

    /// version of the string as the  
    localization key, like this:

    ///

    ///     Text("\_Please visit our  
    [website](<https://www.example.com>).\_")

    ///

    /// When viewed in a Spanish locale,  
    the view uses the Spanish text from the  
    /// strings file, applying the  
    Markdown styling.

    ///

    /// ! [A text view that says Visita  
    nuestro sitio web, with all text

        /// displayed in italics. The words  
        sitio web are colored blue to indicate  
        /// they are a link.] ([SwiftUI-Text-  
        init-localized.png](#))

    ///

    /// > Important: `Text` doesn't  
    render all styling possible in Markdown.  
    It

        /// doesn't support line breaks, soft  
        breaks, or any style of paragraph- or

        /// block-based formatting like  
        lists, block quotes, code blocks, or  
        tables.

        /// It also doesn't support the

        ///

<[doc://com.apple.documentation/documentation/Foundation/AttributeScopes/](https://com.apple.documentation/documentation/Foundation/AttributeScopes/FoundationAttributes/3796122-imageURL)  
[FoundationAttributes/3796122-imageURL](https://com.apple.documentation/documentation/Foundation/AttributeScopes/3796122-imageURL)>

        /// attribute. Parsing with SwiftUI

```
treats any whitespace in the Markdown
    /// string as described by the
    ///
<doc://com.apple.documentation/documentation/Foundation/AttributedString/
MarkdownParsingOptions/
InterpretedSyntax/
inlineOnlyPreservingWhitespace>
    /// parsing option.
    ///
    /// - Parameters:
    ///   - key: The key for a string in
the table identified by `tableName`.
    ///   - tableName: The name of the
string table to search. If `nil`, use the
    ///   table in the
`Localizable.strings` file.
    ///   - bundle: The bundle containing
the strings file. If `nil`, use the
    ///   main bundle.
    ///   - comment: Contextual
information about this key-value pair.
public init(_ key:
LocalizedStringKey, tableName: String? =
nil, bundle: Bundle? = nil, comment:
StaticString? = nil)
}

@available(iOS 16.0, macOS 13, tvOS 16.0,
watchOS 9.0, *)
extension Text {

    /// Creates a text view that displays
```

```
a localized string resource.

    /**
     /// Use this initializer to display a
     localized string that is
     /// represented by a
<doc://com.apple.documentation/documentation/Foundation/LocalizedStringResource>
    /**
     ///     var object =
LocalizedStringResource("pencil")
    ///     Text(object) // Localizes the
resource if possible, or displays
"pencil" if not.
    /**
     @available(iOS 16.0, macOS 13, tvOS
16.0, watchOS 9.0, *)
    public init(_ resource:
LocalizedStringResource)
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Text : View {

    /// The type of view representing the
body of this view.
    /**
     /// When you create a custom view,
Swift infers this type from your
     /// implementation of the required
``View/body-swift.property`` property.
     @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
```

```
    public typealias Body = Never
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension Text {

    /// Description of the style used to
    draw the line for
    `StrikethroughStyleAttribute`  

    /// and `UnderlineStyleAttribute`.
    ///
    /// Use this type to specify
    `underlineStyle` and `strikethroughStyle`
    /// SwiftUI attributes of an
    `AttributedString`.

    public struct LineStyle : Hashable,
Sendable {

        /// Creates a line style.
        ///
        /// - Parameters:
        ///   - pattern: The pattern of
        the line.
        ///   - color: The color of the
        line. If not provided, the foreground
        ///   color of text is used.
        public init(pattern:
Text.LineStyle.Pattern = .solid, color:
Color? = nil)

        /// The pattern, that the line
        has.
```

```
public struct Pattern : Sendable
{
    /// Draw a solid line.
    public static let solid:
Text.LineStyle.Pattern

    /// Draw a line of dots.
    public static let dot:
Text.LineStyle.Pattern

    /// Draw a line of dashes.
    public static let dash:
Text.LineStyle.Pattern

    public static let dashDot:
Text.LineStyle.Pattern

    /// Draw a line of
alternating dashes and two dots.
    public static let dashDotDot:
Text.LineStyle.Pattern
}

/// Draw a single solid line.
public static let single:
Text.LineStyle

    /// 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:
Text.LineStyle, b: Text.LineStyle) ->
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 }
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension Text {
```

```
    /// Creates a text view that displays
the formatted representation
    /// of a reference-convertible value.
    ///
    /// Use this initializer to create a
text view that formats `subject`
    /// using `formatter`.
    /// - Parameters:
    ///   - subject: A
    ///
<doc://com.apple.documentation/documentation/Foundation/ReferenceConvertible>
    /// instance compatible with
`formatter`.
    ///   - formatter: A
    ///
<doc://com.apple.documentation/documentation/Foundation/Formatter>
    /// capable of converting `subject`
into a string representation.
public init<Subject>(_ subject:
Subject, formatter: Formatter) where
Subject : ReferenceConvertible

    /// Creates a text view that displays
the formatted representation
    /// of a Foundation object.
    ///
    /// Use this initializer to create a
text view that formats `subject`
    /// using `formatter`.
    /// - Parameters:
```

```
    /// - subject: An
    ///
<doc://com.apple.documentation/documentation/ObjectiveC/NSObject>
    /// instance compatible with
`formatter`.
    /// - formatter: A
    ///
<doc://com.apple.documentation/documentation/Foundation/Formatter>
    /// capable of converting `subject`
into a string representation.
public init<Subject>(_ subject:
Subject, formatter: Formatter) where
Subject : NSObject
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension Text {

    /// Creates a text view that displays
the formatted representation
    /// of a nonstring type supported by
a corresponding format style.
    ///
    /// Use this initializer to create a
text view backed by a nonstring
    /// value, using a
    ///
<doc://com.apple.documentation/documentation/Foundation/FormatStyle>
    /// to convert the type to a string
```

```
representation. Any changes to the value
    /// update the string displayed by
the text view.
    /**
     /// In the following example, three
``Text`` views present a date with
    /// different combinations of date
and time fields, by using different
    /**
<doc://com.apple.documentation/documentation/Foundation/Date/FormatStyle>
    /// options.
    /**
     ///      @State private var myDate =
Date()
    ///      var body: some View {
    ///          VStack {
    ///              Text(myDate, format:
Date.FormatStyle(date: .numeric,
time: .omitted))
    ///              Text(myDate, format:
Date.FormatStyle(date: .complete,
time: .complete))
    ///              Text(myDate, format:
Date.FormatStyle().hour(.defaultDigitsNoA
MPM).minute())
    ///          }
    ///      }
    /**
     /// ! [Three vertically stacked text
views showing the date with different
    /// levels of detail: 4/1/1976; April
1, 1976; Thursday, April 1,
```

```
    /// 1976.] (Text-init-format-1)
    ///
    /// - Parameters:
    ///   - input: The underlying value
    to display.
    ///   - format: A format style of
    type `F` to convert the underlying value
    ///   of type `F.FormatInput` to a
    string representation.
    public init<F>(_ input:
F.FormatInput, format: F) where F :
FormatStyle, F.FormatInput : Equatable,
F.FormatOutput == String
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Text {

    /// Creates a text view that displays
    the formatted representation
    /// of a nonstring type supported by
    a corresponding format style.
    ///
    /// Use this initializer to create a
    text view backed by a nonstring
    /// value, using a
    ///
<doc://com.apple.documentation/documentation/Foundation/FormatStyle>
    /// to convert the type to an
    attributed string representation. Any
    changes to the value
```

```
    /// update the string displayed by
the text view.
    /**
     /// In the following example, three
``Text`` views present a date with
     /// different combinations of date
and time fields, by using different
    /**
<doc://com.apple.documentation/documentation/Foundation/Date/FormatStyle>
    /// options.
    /**
     ///      @State private var myDate =
Date()
    /**
        var body: some View {
    /**
            VStack {
    /**
                Text(myDate, format:
Date.FormatStyle(date: .numeric,
time: .omitted).attributedStyle)
    /**
                    Text(myDate, format:
Date.FormatStyle(date: .complete,
time: .complete).attributedStyle)
    /**
                    Text(myDate, format:
Date.FormatStyle().hour(.defaultDigitsNoA
MPM).minute().attributedStyle)
    /**
                    }
    /**
            }
    /**
    /**
        /// ! [Three vertically stacked text
views showing the date with different
        /// levels of detail: 4/1/1976; April
1, 1976; Thursday, April 1,
        /// 1976.] (Text-init-format-1)
```

```
    /**
     * - Parameters:
     *   - input: The underlying value
     *     to display.
     *   - format: A format style of
     *     type `F` to convert the underlying value
     *     of type `F.FormatInput` to an
     *     attributed string representation.
     */
    public init<F>(_ input:
F.FormatInput, format: F) where F :
FormatStyle, F.FormatInput : Equatable,
F.FormatOutput == NSAttributedString
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension Text {

    /**
     * Creates an instance that wraps an
     * `Image`, suitable for concatenating
     * with other `Text`
     */
    @available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
    public init(_ image: Image)
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension Text {

    /**
     * A predefined style used to
     * display a `Date`.
     */
    public struct DateStyle : Sendable {
```

```
    /// A style displaying only the
time component for a date.
    /**
     * Text(event.startDate,
style: .time)
    /**
     * Example output:
     * 11:23PM
public static let time:
Text.DateStyle
```

```
    /// A style displaying a date.
    /**
     * Text(event.startDate,
style: .date)
    /**
     * Example output:
     * June 3, 2019
public static let date:
Text.DateStyle
```

```
    /// A style displaying a date as
relative to now.
    /**
     * Text(event.startDate,
style: .relative)
    /**
     * Example output:
     * 2 hours, 23 minutes
     * 1 year, 1 month
public static let relative:
Text.DateStyle
```

```
    /// A style displaying a date as
offset from now.
    /**
     * Text(event.startDate,
style: .offset)
    /**
     * Example output:
     * +2 hours
     * -3 months
public static let offset:
Text.DateStyle
```

```
    /// A style displaying a date as
timer counting from now.
    /**
     * Text(event.startDate,
style: .timer)
    /**
     * Example output:
     * 2:32
     * 36:59:01
public static let timer:
Text.DateStyle
}
```

```
    /// Creates an instance that displays
localized dates and times using a
specific style.
    /**
     * - Parameters:
     *   - date: The target date to
display.
```

```
    /// - style: The style used when
    // displaying a date.
    public init(_ date: Date, style:
Text.DateStyle)

    /// Creates an instance that displays
    // a localized range between two dates.
    ///
    /// - Parameters:
    ///   - dates: The range of dates
    // to display
    public init(_ dates:
ClosedRange<Date>)

    /// Creates an instance that displays
    // a localized time interval.
    ///
    /// Text(DateInterval(start:
event.startDate, duration:
event.duration))
    ///
    /// Example output:
    ///   9:30AM – 3:30PM
    ///
    /// - Parameters:
    ///   - interval: The date interval
    // to display
    public init(_ interval: DateInterval)
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension Text {
```

```
    /// Creates an instance that displays
    a timer counting within the provided
    /// interval.
    ///
    ///     Text(
    ///         timerInterval:
Date.now...Date(timeInterval: 12 * 60,
since: .now))
    ///             pauseTime: Date.now + (10
* 60))
    ///
    /// The example above shows a text
that displays a timer counting down
    /// from "12:00" and will pause when
reaching "10:00".
    ///
    /// - Parameters:
    ///     - timerInterval: The interval
between where to run the timer.
    ///     - pauseTime: If present, the
date at which to pause the timer.
    ///             The default is `nil`
which indicates to never pause.
    ///     - countsDown: Whether to
count up or down. The default is `true`.
    ///     - showsHours: Whether to
include an hours component if there are
    ///             more than 60 minutes left
on the timer. The default is `true`.
public init(timerInterval:
ClosedRange<Date>, pauseTime: Date? =
nil, countsDown: Bool = true, showsHours:
```

```
Bool = true)
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension Text {

    /// Creates a text view that displays
    styled attributed content.
    ///
    /// Use this initializer to style
    text according to attributes found in the
    specified
    ///
<doc://com.apple.documentation/documentation/Foundation/AttributedString>.
    /// Attributes in the attributed
    string take precedence over styles added
    by
    /// view modifiers. For example, the
    attributed text in the following
    /// example appears in blue, despite
    the use of the
``View/foregroundColor(_:)``
    /// modifier to use red throughout
    the enclosing ``VStack``:
    ///
    ///     var content: NSAttributedString
{
    ///         var attributedString =
    NSAttributedString("Blue text")
    ///
    attributedString.foregroundColor = .blue
```

```
    ///         return attributedString
    ///     }
    ///
    ///     var body: some View {
    ///         VStack {
    ///             Text(content)
    ///             Text("Red text")
    ///         }
    ///         .foregroundColor(.red)
    ///     }
    ///
    ///     !-[A vertical stack of two text
views, the top labeled Blue Text with a
/// blue font color, and the bottom
labeled Red Text with a red font
/// color.] (SwiftUI-Text-init-
attributed.png)
    ///
    /// SwiftUI combines text attributes
with SwiftUI modifiers whenever
    /// possible. For example, the
following listing creates text that is
    /// both bold and red:
    ///
    ///     var content: NSAttributedString
{
    ///         var content =
AttributedString("Some text")
    ///
content.inlinePresentationIntent
= .stronglyEmphasized
    ///         return content
    /// }
```

```
///  
///     var body: some View {  
///  
Text(content).foregroundColor(Color.red)  
///     }  
///  
/// A SwiftUI ``Text`` view renders  
most of the styles defined by the  
/// Foundation attribute  
///  
<doc://com.apple.documentation/documentation/Foundation/AttributeScopes/  
FoundationAttributes/3796123-  
inlinePresentationIntent>, like the  
///  
<doc://com.apple.documentation/documentation/Foundation/InlinePresentationIntent/  
3746899-stronglyEmphasized>  
/// value, which SwiftUI presents as  
bold text.  
///  
/// > Important: ``Text`` uses only a  
subset of the attributes defined in  
///  
<doc://com.apple.documentation/documentation/Foundation/AttributeScopes/  
FoundationAttributes>.  
/// `Text` renders all  
///  
<doc://com.apple.documentation/documentation/Foundation/InlinePresentationIntent>  
/// attributes except for  
///
```

```
<doc://com.apple.documentation/documentation/Foundation/InlinePresentationIntent/3787563-lineBreak> and
    /**
<doc://com.apple.documentation/documentation/Foundation/InlinePresentationIntent/3787564-softBreak>.
    /**
     It also renders the
    /**
<doc://com.apple.documentation/Foundation/AttributeScopes/FoundationAttributes/3764633-link>
    /**
     attribute as a clickable link.
`Text` ignores any other
    /**
     Foundation-defined attributes in
an attributed string.
    /**
    /**
     SwiftUI also defines additional
attributes in the attribute scope
    /**
<doc://com.apple.documentation/documentation/Foundation/AttributeScopes/SwiftUIAttributes>
    /**
     which you can access from an
attributed string's
    /**
<doc://com.apple.documentation/documentation/Foundation/AttributeScopes/3788543-swiftUI>
    /**
     property. SwiftUI attributes take
precedence over equivalent attributes
    /**
     from other frameworks, such as
    /**

```

```
<doc://com.apple.documentation/documentation/Foundation/AttributeScopes/UIKitAttributes> and
    /**
<doc://com.apple.documentation/documentation/Foundation/AttributeScopes/AppKitAttributes>.

    /**
    /**
    /**
     * You can create an
`AttributedString` with Markdown syntax,
which allows
    /**
        you to style distinct runs within
a `Text` view:
    /**
        let content = try!
AttributedString(
    /**
        markdown: "##Thank You!##
Please visit our [website]
(http://example.com).")
    /**
    /**
        var body: some View {
    /**
        Text(content)
    /**
}
    /**
    /**
        The `##` syntax around "Thank
You!" applies an
    /**
<doc://com.apple.documentation/documentation/Foundation/AttributeScopes/FoundationAttributes/3796123-
inlinePresentationIntent>
    /**
        attribute with the value
```

```
///  
<doc://com.apple.documentation/documentation/Foundation/InlinePresentationIntent/3746899-stronglyEmphasized>  
    /// SwiftUI renders this as  
    /// bold text, as described earlier.  
The link syntax around "website"  
    /// creates a  
    ///  
<doc://com.apple.documentation/documentation/Foundation/AttributeScopes/FoundationAttributes/3764633-link>  
    /// attribute, which `Text` styles to indicate it's a link; by default,  
    /// clicking or tapping the link opens the linked URL in the user's default  
    /// browser. Alternatively, you can perform custom link handling by putting  
    /// an ``OpenURLAction`` in the text view's environment.  
    ///  
    /// ![A text view that says Thank you. Please visit our website. The text  
    /// The view displays the words Thank you in a bold font, and the word  
    /// website styled to indicate it is a  
    /// link.]({SwiftUI-Text-init-markdown.png})  
    ///  
    /// You can also use Markdown syntax in localized string keys, which means
```

```
    /// you can write the above example
    /// without needing to explicitly create
    /// an `AttributedString`:
    ///
    ///     var body: some View {
    ///         Text("**Thank You!**")
    ///         Please visit our [website]
    ///         (https://example.com).")
    ///     }
    ///
    /// In your app's strings files, use
    /// Markdown syntax to apply styling
    /// to the app's localized strings.
    You also use this approach when you want
    /// to perform automatic grammar
    agreement on localized strings, with
    /// the `^-[text](inflect:true)`
    syntax.
    ///
    /// For details about Markdown syntax
    support in SwiftUI, see
    ///
``Text/init(_:tableName:bundle:comment:)``
```
    ///
    /// - Parameters:
    ///   - attributedContent: An
    ///     attributed string to style and display,
    ///     in accordance with its
    attributes.
    public init(_ attributedContent:
    AttributedString)
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Text {  
  
    /// Concatenates the text in two text  
    views in a new text view.  
    ///  
    /// - Parameters:  
    ///     - lhs: The first text view with  
    text to combine.  
    ///     - rhs: The second text view  
    with text to combine.  
    ///  
    /// - Returns: A new text view  
    containing the combined contents of the  
    two  
    /// input text views.  
    public static func + (lhs: Text, rhs:  
    Text) -> Text  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Text {  
  
    /// The type of truncation to apply  
    to a line of text when it's too long to  
    /// fit in the available space.  
    ///  
    /// When a text view contains more  
    text than it's able to display, the view  
    /// might truncate the text and place
```

```
an ellipsis (...) at the truncation
    /// point. Use the
``View/truncationMode(_:)`` modifier with
one of the
    /// `TruncationMode` values to
indicate which part of the text to
    /// truncate, either at the
beginning, in the middle, or at the end.
public enum TruncationMode : Sendable
{
    /// Truncate at the beginning of
the line.
    ///
    /// Use this kind of truncation
to omit characters from the beginning of
    /// the string. For example, you
could truncate the English alphabet as
    /// "...wxyz".
    case head

    /// Truncate at the end of the
line.
    ///
    /// Use this kind of truncation
to omit characters from the end of the
    /// string. For example, you
could truncate the English alphabet as
    /// "abcd...".
    case tail

    /// Truncate in the middle of the
line.
```

```
    /**
     * Use this kind of truncation
     * to omit characters from the middle of
     * the string. For example, you
     * could truncate the English alphabet as
     * "ab...yz".
     */
    case middle

        /**
         * 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:
Text.TruncationMode, b:
Text.TruncationMode) -> Bool

        /**
         * 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)
```

```
    /// 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 scheme for transforming the  
capitalization of characters within text.
```

```
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)
```

```
public enum Case : Sendable {
```

```
    /// Displays text in all  
uppercase characters.
```

```
    ///
```

```
    /// For example, "Hello" would be  
displayed as "HELLO".
```

```
    ///
```

```
    /// - SeeAlso:  
`StringProtocol.uppercased(with:)`  
    case uppercase
```

```
    /// Displays text in all  
lowercase characters.
```

```
    ///
```

```
    /// For example, "Hello" would be  
displayed as "hello".
```

```
    ///
```

```
    /// - SeeAlso:  
`StringProtocol.lowercased(with:)`
```

## case lowercase

```
    /// 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:  
Text.Case, b: Text.Case) -> Bool
```

```
    /// 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)  
  
    /// 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 }
```

```
        }
    }

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Text {

    /// Specifies the language for
    typesetting.
    ///
    /// In some cases `Text` may contain
    text of a particular language which
    /// doesn't match the device UI
    language. In that case it's useful to
    /// specify a language so line
    height, line breaking and spacing will
    /// respect the script used for that
    language. For example:
    ///
    ///     Text(verbatim: "ແອນເປົລ")
    ///             .typesettingLanguage(.ini
    t(languageCode: .thai))
    ///
    /// Note: this language does not
    affect text localization.
    ///
    /// - Parameters:
    ///   - language: The explicit
    language to use for typesetting.
    ///   - isEnabled: A Boolean value
    that indicates whether text language is
    ///     added
    /// - Returns: Text with the
```

```
typesetting language set to the value you
    /// supply.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public func typesettingLanguage(_
language: Locale.Language, isEnabled:
Bool = true) -> Text

    /// Specifies the language for
typesetting.
    ///
    /// In some cases `Text` may contain
text of a particular language which
    /// doesn't match the device UI
language. In that case it's useful to
    /// specify a language so line
height, line breaking and spacing will
    /// respect the script used for that
language. For example:
    ///
    ///     Text(verbatim:
"ແອນເປົລ").typesettingLanguage(
    ///             .explicit(.init(languageC
ode: .thai)))
    ///
    /// Note: this language does not
affect text localized localization.
    ///
    /// - Parameters:
    ///   - language: The language to use
for typesetting.
    ///   - isEnabled: A Boolean value
that indicates whether text language is
```

```
    ///      added
    /// - Returns: Text with the
    typesetting language set to the value you
    ///      supply.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public func typesettingLanguage(_
language: TypesettingLanguage, isEnabled:
Bool = true) -> Text
}

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

    /// Adds a custom attribute to the
text view.
    ///
    /// Only one attribute of each type
may be attached to each text
    /// view, with inner attributes
taking precedence.
    ///
    /// - Parameter value: the attribute
to attach.
    ///
    /// - Returns: a version of the text
view with `value` attached.
    public func customAttribute<T>(_
value: T) -> Text where T : TextAttribute
}

@available(iOS 17.0, macOS 14.0, tvOS
```

```
17.0, watchOS 10.0, *)
extension Text {

    /// A value describing the layout and
    custom attributes of a tree
    /// of `Text` views.
    public struct Layout : RandomAccessCollection, Equatable {

        /// Indicates if this text is
        truncated.
        @available(iOS 18.0, macOS 15.0,
        tvOS 18.0, watchOS 11.0, visionOS 2.0, *)
            public var isTruncated: Bool {
        get }

        /// The position of the first
        element in a nonempty collection.
        ///
        /// If the collection is empty,
        `startIndex` is equal to `endIndex`.
        public var startIndex: Int {
        get }

        /// The collection's "past the
        end" position---that is, the position one
        /// greater than the last valid
        subscript argument.
        ///
        /// When you need a range that
        includes the last element of a
        collection, use
        /// the half-open range operator
```

(`..<`) with `endIndex`. The `..<` operator

    /// creates a range that doesn't include the upper bound, so it's always  
    /// safe to use with `endIndex`.

For example:

```
    ///  
    ///     let numbers = [10, 20,  
30, 40, 50]  
    ///     if let index =  
numbers.firstIndex(of: 30) {  
        ///  
print(numbers[index ..<  
numbers.endIndex])  
        ///     }  
        ///     // Prints "[30, 40, 50]"  
        ///  
        /// If the collection is empty,  
`endIndex` is equal to `startIndex`.  
    public var endIndex: Int { get }
```

    /// Accesses the element at the specified position.

```
    ///  
    /// The following example  
accesses an element of an array through its
```

```
    /// subscript to print its value:  
    ///  
    ///     var streets = ["Adams",  
"Bryant", "Channing", "Douglas",  
"Evarts"]  
    ///     print(streets[1])
```

```
    ///      // Prints "Bryant"
    ///
    /// You can subscript a
collection with any valid index other
than the
        /// collection's end index. The
end index refers to the position one past
        /// the last element of a
collection, so it doesn't correspond with
an
        /// element.
        ///
        /// - Parameter position: The
position of the element to access.
`position`
        /// must be a valid index of
the collection that is not equal to the
        /// `endIndex` property.
        ///
        /// - Complexity: O(1)
public subscript(index: Int) ->
Text.Layout.Line { get }

        /// The index of a character in
the source text. An opaque
        /// type, this is intended to be
used to determine relative
        /// locations of elements in the
layout, rather than how they
        /// map to the source strings.
@frozen public struct
CharacterIndex : Comparable, Hashable,
Strideable, Sendable {
```

```
    /// Returns a Boolean value  
indicating whether the value of the first  
    /// argument is less than  
that of the second argument.
```

```
    ///  
    /// This function is the only  
requirement of the `Comparable` protocol.  
The
```

```
    /// remainder of the  
relational operator functions are  
implemented by the
```

```
    /// standard library for any  
type that conforms to `Comparable`.
```

```
    ///  
    /// - Parameters:  
    /// - lhs: A value to  
compare.
```

```
    /// - rhs: Another value to  
compare.
```

```
public static func <(lhs:  
Text.Layout.CharacterIndex, rhs:  
Text.Layout.CharacterIndex) -> Bool
```

```
    /// Returns a value that is  
offset the specified distance from this  
value.
```

```
    ///  
    /// Use the `advanced(by:)`  
method in generic code to offset a value  
by a
```

```
    /// specified distance. If  
you're working directly with numeric
```

values, use

```
    /// the addition operator
(`+`) instead of this method.
    ///
    /// func addOne<T:
Strideable>(to x: T) -> T
    /// where T.Stride:
ExpressibleByIntegerLiteral
    ///
    {
    ///
        return
x.advanced(by: 1)
    ///
    }
    ///
    /// let x = addOne(to: 5)
    /// // x == 6
    /// let y = addOne(to:
```

3.5)

```
    ///
    // y = 4.5
    ///
    /// If this type's `Stride`
type conforms to `BinaryInteger`, then
for a
```

```
    /// value `x`, a distance
`n`, and a value `y = x.advanced(by: n)`,
    /// `x.distance(to: y) == n`.
```

Using this method with types that have a  
    /// noninteger `Stride` may  
result in an approximation. If the result  
of

```
    /// advancing by `n` is not
representable as a value of this type,
then a
```

```
    /// runtime error may occur.
```

```
    /**
     * - Parameter n: The
     * distance to advance this value.
     * - Returns: A value that
     * is offset from this value by `n`.
     */
    /**
     * - Complexity: O(1)
     public func advanced(by n:
Int) -> Text.Layout.CharacterIndex

        /**
         * Returns the distance from
         * this value to the given value, expressed
         * as a
         * stride.
         */
        /**
         * If this type's `Stride`
         * type conforms to `BinaryInteger`, then
         * for two
         *   /**
         *   values `x` and `y`, and a
         *   distance `n = x.distance(to: y)`,
         *   /**
         *   `x.advanced(by: n) == y`.
         * Using this method with types that have a
         *   /**
         *   noninteger `Stride` may
         * result in an approximation.
         */
        /**
         * - Parameter other: The
         * value to calculate the distance to.
         * - Returns: The distance
         * from this value to `other`.
         */
        /**
         * - Complexity: O(1)
         public func distance(to
other: Text.Layout.CharacterIndex) -> Int
```

```
    /// 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)
```

```
    /// A type that represents  
the distance between two values.  
    @available(iOS 17.0, tvOS  
17.0, watchOS 10.0, macOS 14.0, *)  
    public typealias Stride = Int  
  
    /// 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 }  
    /// The typographic bounds of an  
element in a text layout.  
    @frozen public struct  
TypographicBounds : Equatable, Sendable {
```

```
    /// The position of the left  
    edge of the element's  
    /// baseline, relative to the  
    text view.  
    public var origin: CGPoint  
  
    /// The width of the element.  
    public var width: CGFloat  
  
    /// The ascent of the  
    element.  
    public var ascent: CGFloat  
  
    /// The descent of the  
    element.  
    public var descent: CGFloat  
  
    /// The leading of the  
    element.  
    public var leading: CGFloat  
  
    /// Initializes to an empty  
    bounds with zero origin.  
    public init()  
  
        /// Returns a rectangle  
        encapsulating the bounds.  
        public var rect: CGRect { get  
    }  
  
        /// Returns a Boolean value  
        indicating whether two values are equal.  
        ///
```

```
        /// Equality is the inverse  
of inequality. For any values `a` and  
`b`,  
`a != b` is `false`.  
        ///  
        /// - Parameters:  
        ///   - lhs: A value to  
compare.  
        ///   - rhs: Another value to  
compare.  
    public static func == (a:  
Text.Layout.TypographicBounds, b:  
Text.Layout.TypographicBounds) -> Bool  
}
```

  

```
        /// A single line in a text  
layout: a collection of runs of  
        /// placed glyphs.  
    public struct Line :  
RandomAccessCollection, Equatable {  
  
        /// The origin of the line.  
    public var origin: CGPoint  
  
        /// The position of the first  
element in a nonempty collection.  
        ///  
        /// If the collection is  
empty, `startIndex` is equal to  
`endIndex`.  
    public var startIndex: Int {  
get }
```

```
        /// The collection's "past  
the end" position---that is, the position  
one
```

```
        /// greater than the last  
valid subscript argument.
```

```
        ///
```

```
        /// When you need a range  
that includes the last element of a  
collection, use
```

```
        /// the half-open range  
operator (`..<`) with `endIndex`. The  
`..<` operator
```

```
        /// creates a range that  
doesn't include the upper bound, so it's  
always
```

```
        /// safe to use with  
`endIndex`. For example:
```

```
        ///
```

```
        ///     let numbers = [10,  
20, 30, 40, 50]
```

```
        ///     if let index =  
numbers.firstIndex(of: 30) {
```

```
        ///
```

```
print(numbers[index ..<  
numbers.endIndex])
```

```
        /// }
```

```
        /// // Prints "[30, 40,  
50]"
```

```
        ///
```

```
        /// If the collection is  
empty, `endIndex` is equal to  
`startIndex`.
```

```
        public var endIndex: Int {  
get }  
  
        /// Accesses the element at  
the specified position.  
        ////  
        /// The following example  
accesses an element of an array through  
its  
        /// subscript to print its  
value:  
        ////  
        ///     var streets =  
["Adams", "Bryant", "Channing",  
"Douglas", "Evarts"]  
        ///     print(streets[1])  
        ///     // Prints "Bryant"  
        ////  
        /// You can subscript a  
collection with any valid index other  
than the  
        /// collection's end index.  
The end index refers to the position one  
past  
        /// the last element of a  
collection, so it doesn't correspond with  
an  
        /// element.  
        ////  
        /// - Parameter position: The  
position of the element to access.  
`position`  
        ///     must be a valid index
```

of the collection that is not equal to the

```
    /// `endIndex` property.  
    ///  
    /// - Complexity: O(1)  
    public subscript(index: Int)  
-> Text.Layout.Run { get }
```

/// The typographic bounds of the line.

```
        public var typographicBounds:  
Text.Layout.TypographicBounds { get }
```

/// 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 == (lhs:  
Text.Layout.Line, rhs: Text.Layout.Line)  
-> Bool
```

/// A type representing the sequence's elements.

```
        @available(iOS 17.0, tvOS  
17.0, watchOS 10.0, macOS 14.0, *)  
            public typealias Element =  
Text.Layout.Run  
  
                /// A type that represents a  
position in the collection.  
                ///  
                /// Valid indices consist of  
the position of every element and a  
                /// "past the end" position  
that's not valid for use as a subscript  
                /// argument.  
        @available(iOS 17.0, tvOS  
17.0, watchOS 10.0, macOS 14.0, *)  
            public typealias Index = Int  
  
                /// A type that represents  
the indices that are valid for  
subscripting the  
                /// collection, in ascending  
order.  
        @available(iOS 17.0, tvOS  
17.0, watchOS 10.0, macOS 14.0, *)  
            public typealias Indices =  
Range<Int>  
  
                /// A type that provides the  
collection's iteration interface and  
                /// encapsulates its  
iteration state.  
                ///  
                /// By default, a collection
```

```
conforms to the `Sequence` protocol by
    /// supplying
`IndexingIterator` as its associated
`Iterator`
    /// type.
@available(iOS 17.0, tvOS
17.0, watchOS 10.0, macOS 14.0, *)
public typealias Iterator =
IndexingIterator<Text.Layout.Line>

        /// A collection representing
a contiguous subrange of this
collection's
        /// elements. The subsequence
shares indices with the original
collection.
        ///
        /// The default subsequence
type for collections that don't define
their own
        /// is `Slice`.
@available(iOS 17.0, tvOS
17.0, watchOS 10.0, macOS 14.0, *)
public typealias SubSequence
= Slice<Text.Layout.Line>
}

        /// A run of placed glyphs in a
text layout.
public struct Run :
RandomAccessCollection, Equatable {

        /// The position of the first
```

element in a nonempty collection.

///

/// If the collection is empty, `startIndex` is equal to ` endIndex`.

```
public var startIndex: Int {  
get }
```

/// The collection's "past the end" position---that is, the position one

/// greater than the last valid subscript argument.

///

/// When you need a range that includes the last element of a collection, use

/// the half-open range operator (`..<`) with `endIndex`. The `..<` operator

/// creates a range that doesn't include the upper bound, so it's always

/// safe to use with `endIndex`. For example:

///

```
/// let numbers = [10,  
20, 30, 40, 50]
```

```
/// if let index =  
numbers.firstIndex(of: 30) {
```

///

```
print(numbers[index ..<  
numbers.endIndex])
```

```
        ///      }
        ///      // Prints "[30, 40,
50]"
        ///
        /// If the collection is
empty, `endIndex` is equal to
`startIndex`.
    public var endIndex: Int {
get }

        /// Accesses the element at
the specified position.
        ///
        /// The following example
accesses an element of an array through
its
        /// subscript to print its
value:
        ///
        ///     var streets =
["Adams", "Bryant", "Channing",
"Douglas", "Evarts"]
        ///     print(streets[1])
        ///     // Prints "Bryant"
        ///
        /// You can subscript a
collection with any valid index other
than the
        /// collection's end index.
The end index refers to the position one
past
        /// the last element of a
collection, so it doesn't correspond with
```

an

```
    /// element.  
    ///  
    /// - Parameter position: The  
position of the element to access.  
`position`  
        /// must be a valid index  
of the collection that is not equal to  
the  
        /// `endIndex` property.  
        ///  
        /// - Complexity: O(1)  
    public subscript(index: Int)  
-> Text.Layout.RunSlice { get }  
  
        /// Accesses a contiguous  
subrange of the collection's elements.  
        ///  
        /// The accessed slice uses  
the same indices for the same elements as  
the  
        /// original collection uses.  
Always use the slice's `startIndex`  
property  
        /// instead of assuming that  
its indices start at a particular value.  
        ///  
        /// This example demonstrates  
getting a slice of an array of strings,  
finding  
        /// the index of one of the  
strings in the slice, and then using that  
index
```

```
        /// in the original array.  
        ///  
        ///     let streets =  
["Adams", "Bryant", "Channing",  
"Douglas", "Evarts"]  
        ///     let streetsSlice =  
streets[2 ..< streets.endIndex]  
        ///     print(streetsSlice)  
        ///     // Prints  
" ["Channing", "Douglas", "Evarts"]"  
        ///  
        ///     let index =  
streetsSlice.firstIndex(of:  
"Evarts")    // 4  
        ///  
print(streets[index!])  
        ///     // Prints "Evarts"  
        ///  
        /// - Parameter bounds: A  
range of the collection's indices. The  
bounds of  
        ///     the range must be valid  
indices of the collection.  
        ///  
        /// - Complexity: O(1)  
public subscript(bounds:  
Range<Int>) -> Text.Layout.RunSlice { get  
}  
  
        /// The custom attribute of  
type `T` associated with the  
        /// run of glyphs, or nil.  
public subscript<T>(key:
```

```
T.Type) -> T? where T : TextAttribute {  
get }  
  
        /// The layout direction of  
the text run.  
        public var layoutDirection:  
LayoutDirection { get }  
  
        /// The typographic bounds of  
the run of glyphs.  
        public var typographicBounds:  
Text.Layout.TypographicBounds { get }  
  
        /// The array of character  
indices corresponding to the  
        /// glyphs in `self`.  
        public var characterIndices:  
[Text.Layout.CharacterIndex] { get }  
  
        /// 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 == (lhs:  
Text.Layout.Run, rhs: Text.Layout.Run) ->  
Bool  
  
        /// A type representing the  
sequence's elements.  
        @available(iOS 17.0, tvOS  
17.0, watchOS 10.0, macOS 14.0, *)  
        public typealias Element =  
Text.Layout.RunSlice  
  
        /// A type that represents a  
position in the collection.  
        ///  
        /// Valid indices consist of  
the position of every element and a  
        /// "past the end" position  
that's not valid for use as a subscript  
        /// argument.  
        @available(iOS 17.0, tvOS  
17.0, watchOS 10.0, macOS 14.0, *)  
        public typealias Index = Int  
  
        /// A type that represents  
the indices that are valid for  
subscripting the  
        /// collection, in ascending  
order.  
        @available(iOS 17.0, tvOS  
17.0, watchOS 10.0, macOS 14.0, *)  
        public typealias Indices =  
Range<Int>
```

```
        /// A type that provides the
collection's iteration interface and
        /// encapsulates its
iteration state.
        ///
        /// By default, a collection
conforms to the `Sequence` protocol by
        /// supplying
`IndexingIterator` as its associated
`Iterator`
        /// type.
@available(iOS 17.0, tvOS
17.0, watchOS 10.0, macOS 14.0, *)
public typealias Iterator =
IndexingIterator<Text.Layout.Run>

        /// A collection representing
a contiguous subrange of this
collection's
        /// elements. The subsequence
shares indices with the original
collection.
        ///
        /// The default subsequence
type for collections that don't define
their own
        /// is `Slice`.
@available(iOS 17.0, tvOS
17.0, watchOS 10.0, macOS 14.0, *)
public typealias SubSequence
= Text.Layout.RunSlice
}
```

```
    /// A slice of a run of placed
    glyphs in a text layout.
    public struct RunSlice :  
RandomAccessCollection, Equatable {  
  
    public var run:  
Text.Layout.Run  
  
        /// The indices that are
valid for subscripting the collection, in
ascending
        /// order.  
        ///  
        /// A collection's `indices`
property can hold a strong reference to
the
        /// collection itself,
causing the collection to be nonuniquely
referenced.  
        /// If you mutate the
collection while iterating over its
indices, a strong
        /// reference can result in
an unexpected copy of the collection. To
avoid
        /// the unexpected copy, use
the `index(after:)` method starting with
        /// `startIndex` to produce
indices instead.  
        ///  
        ///     var c =
MyFancyCollection([10, 20, 30, 40, 50])
        ///     var i = c.startIndex
```

```
        ///      while i != c.endIndex
{
    ///          c[i] /= 5
    ///          i =
c.index(after: i)
    ///      }
    ///      // c ==
MyFancyCollection([2, 4, 6, 8, 10])
public var indices:
Range<Int>

public init(run:
Text.Layout.Run, indices: Range<Int>)

        /// The position of the first
element in a nonempty collection.
        ///
        /// If the collection is
empty, `startIndex` is equal to
`endIndex`.
public var startIndex: Int {
get }

        /// The collection's "past
the end" position---that is, the position
one
        /// greater than the last
valid subscript argument.
        ///
        /// When you need a range
that includes the last element of a
collection, use
        /// the half-open range
```

```
operator (`..<`) with `endIndex`. The
`..<` operator
    /// creates a range that
    doesn't include the upper bound, so it's
    always
        /// safe to use with
`endIndex`. For example:
    /**
     *      let numbers = [10,
20, 30, 40, 50]
     *      if let index =
numbers.firstIndex(of: 30) {
     *          /**
print(numbers[index ..<
numbers.endIndex])
     *          }
     *          // Prints "[30, 40,
50]"
     */
     *      // If the collection is
empty, `endIndex` is equal to
`startIndex`.
public var endIndex: Int {
get }

    /// Accesses the element at
the specified position.
    /**
     *      // The following example
accesses an element of an array through
its
     *      // subscript to print its
value:
```

```
    /**
     *      var streets =
["Adams", "Bryant", "Channing",
"Douglas", "Evarts"]
     *      print(streets[1])
     *      // Prints "Bryant"
     *
     *      // You can subscript a
collection with any valid index other
than the
     *      // collection's end index.
The end index refers to the position one
past
     *      // the last element of a
collection, so it doesn't correspond with
an
     *      // element.
     */
     *      // - Parameter position: The
position of the element to access.
`position`
     *      // must be a valid index
of the collection that is not equal to
the
     *      // `endIndex` property.
     */
     *      // - Complexity: O(1)
public subscript(index: Int)
-> Text.Layout.RunSlice { get }

     *      // Accesses a contiguous
subrange of the collection's elements.
     */
```

```
        /// The accessed slice uses
the same indices for the same elements as
the
        /// original collection uses.
Always use the slice's `startIndex`
property
        /// instead of assuming that
its indices start at a particular value.
        ///
        /// This example demonstrates
getting a slice of an array of strings,
finding
        /// the index of one of the
strings in the slice, and then using that
index
        /// in the original array.
        ///
        ///     let streets =
["Adams", "Bryant", "Channing",
"Douglas", "Evarts"]
        ///     let streetsSlice =
streets[2 ..< streets.endIndex]
        ///     print(streetsSlice)
        ///     // Prints
"["Channing", "Douglas", "Evarts"]"
        ///
        ///     let index =
streetsSlice.firstIndex(of:
"Evarts")    // 4
        ///
print(streets[index!])
        ///     // Prints "Evarts"
        ///
```

```
        /// - Parameter bounds: A
range of the collection's indices. The
bounds of
        /// the range must be valid
indices of the collection.
        ///
        /// - Complexity: O(1)
public subscript(bounds:
Range<Int>) -> Text.Layout.RunSlice { get
}

        /// The custom attribute of
type `T` associated with the
        /// run of glyphs, or nil.
public subscript<T>(key:
T.Type) -> T? where T : TextAttribute {
get }

        /// The typographic bounds of
the partial run of glyphs.
public var typographicBounds:
Text.Layout.TypographicBounds { get }

        /// The array of character
indices corresponding to the
        /// glyphs in `self`.
public var characterIndices:
[Text.Layout.CharacterIndex] { get }

        /// 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:
Text.Layout.RunSlice, b:
Text.Layout.RunSlice) -> Bool

    /// A type representing the
sequence's elements.
@available(iOS 17.0, tvOS
17.0, watchOS 10.0, macOS 14.0, *)
public typealias Element =
Text.Layout.RunSlice

    /// A type that represents a
position in the collection.
///
    /// Valid indices consist of
the position of every element and a
    /// "past the end" position
that's not valid for use as a subscript
    /// argument.
@available(iOS 17.0, tvOS
17.0, watchOS 10.0, macOS 14.0, *)
public typealias Index = Int
```

```
        /// A type that represents
the indices that are valid for
subscripting the
        /// collection, in ascending
order.
    @available(iOS 17.0, tvOS
17.0, watchOS 10.0, macOS 14.0, *)
    public typealias Indices =
Range<Int>

        /// A type that provides the
collection's iteration interface and
        /// encapsulates its
iteration state.
        ///
        /// By default, a collection
conforms to the `Sequence` protocol by
        /// supplying
`IndexingIterator` as its associated
`Iterator`
        /// type.
    @available(iOS 17.0, tvOS
17.0, watchOS 10.0, macOS 14.0, *)
    public typealias Iterator =
IndexingIterator<Text.Layout.RunSlice>

        /// A collection representing
a contiguous subrange of this
collection's
        /// elements. The subsequence
shares indices with the original
collection.
        ///
```

```
        /// The default subsequence
type for collections that don't define
their own
        /// is `Slice`.
@available(iOS 17.0, tvOS
17.0, watchOS 10.0, macOS 14.0, *)
public typealias SubSequence
= Text.Layout.RunSlice
}

        /// 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:
Text.Layout, b: Text.Layout) -> Bool

        /// A type representing the
sequence's elements.
@available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
public typealias Element =
Text.Layout.Line

        /// A type that represents a
```

```
position in the collection.

    /**
     * Valid indices consist of the
     * position of every element and a
     * "past the end" position
     * that's not valid for use as a subscript
     * argument.
     */
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
public typealias Index = Int

    /**
     * A type that represents the
     * indices that are valid for subscripting
     * the
     * collection, in ascending
     * order.
     */
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
public typealias Indices =
Range<Int>

    /**
     * A type that provides the
     * collection's iteration interface and
     * encapsulates its iteration
     * state.
     */
    /**
     * By default, a collection
     * conforms to the `Sequence` protocol by
     * supplying `IndexingIterator`
     * as its associated `Iterator`
     * type.
     */
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
```

```
    public typealias Iterator =  
IndexingIterator<Text.Layout>  
  
        /// A collection representing a  
contiguous subrange of this collection's  
        /// elements. The subsequence  
shares indices with the original  
collection.  
        ////  
        /// The default subsequence type  
for collections that don't define their  
own  
        /// is `Slice`.  
        @available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)  
        public typealias SubSequence =  
Slice<Text.Layout>  
    }  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Text {  
  
    /// A preference key that provides  
the `Text.Layout` values for all  
    /// text views in the queried  
subtree.  
    public struct LayoutKey :  
PreferenceKey, Sendable {  
  
        public struct AnchoredLayout :  
Equatable {
```

```
        /// The origin of the text
layout.
    public var origin:
Anchor<CGPoint>

        /// The text layout value.
    public var layout:
Text.Layout

        /// 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:
Text.LayoutKey.AnchoredLayout, b:
Text.LayoutKey.AnchoredLayout) -> Bool
}

        /// The type of value produced by
this preference.
    public typealias Value =
[Text.LayoutKey.AnchoredLayout]
```

```
    /// The default value of the  
    preference.  
    ///  
    /// Views that have no explicit  
    value for the key produce this default  
    /// value. Combining child views  
    may remove an implicit value produced by  
    /// using the default. This means  
    that `reduce(value: &x, nextValue:  
    /// {defaultValue})` shouldn't  
    change the meaning of `x`.  
    public static let defaultValue:  
Text.LayoutKey.Value
```

```
    /// Combines a sequence of values  
    by modifying the previously-accumulated  
    /// value with the result of a  
    closure that provides the next value.  
    ///  
    /// This method receives its  
    values in view-tree order. Conceptually,  
    this  
    /// combines the preference value  
    from one tree with that of its next  
    /// sibling.  
    ///  
    /// - Parameters:  
    ///     - value: The value  
    accumulated through previous calls to  
    this method.  
    ///     The implementation should  
    modify this value.
```

```
        /// - nextValue: A closure that
returns the next value in the sequence.
    public static func reduce(value:
inout Text.LayoutKey.Value, nextValue: ()-> Text.LayoutKey.Value)
    }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Text {

    /// Sets the color of the text
displayed by this view.
    ///
    /// Use this method to change the
color of the text rendered by a text
view.
    ///
    /// For example, you can display the
names of the colors red, green, and
    /// blue in their respective colors:
    ///
    ///     HStack {
    ///
    ///         Text("Red").foregroundColor(.red)
    ///
    ///         Text("Green").foregroundColor(.green)
    ///
    ///         Text("Blue").foregroundColor(.blue)
    ///     }
    ///
    ///
    /// ! [Three text views arranged
```

```
horizontally, each containing
    ///      the name of a color displayed
in that
    ///      color.](SwiftUI-Text-
foregroundColor.png)
    ///
    /// - Parameter color: The color to
use when displaying this text.
    /// - Returns: A text view that uses
the color value you supply.
    @available(iOS, introduced: 13.0,
deprecated: 100000.0, renamed:
"foregroundStyle(_)")
    @available(macOS, introduced: 10.15,
deprecated: 100000.0, renamed:
"foregroundStyle(_)")
    @available(tvOS, introduced: 13.0,
deprecated: 100000.0, renamed:
"foregroundStyle(_)")
    @available(watchOS, introduced: 6.0,
deprecated: 100000.0, renamed:
"foregroundStyle(_)")
    @available(visionOS, introduced: 1.0,
deprecated: 100000.0, renamed:
"foregroundStyle(_)")
    nonisolated public func
foregroundColor(_ color: Color?) -> Text

    /// Sets the style of the text
displayed by this view.
    ///
    /// Use this method to change the
rendering style of the text
```

```
    /// rendered by a text view.  
    ///  
    /// For example, you can display the  
    names of the colors red,  
    /// green, and blue in their  
    respective colors:  
    ///  
    ///     HStack {  
    ///  
    Text("Red").foregroundStyle(.red)  
    ///  
    Text("Green").foregroundStyle(.green)  
    ///  
    Text("Blue").foregroundStyle(.blue)  
    ///     }  
    ///  
    /// ! [Three text views arranged  
    horizontally, each containing  
    ///      the name of a color displayed  
    in that  
    ///      color.] (SwiftUI-Text-  
    foregroundColor.png)  
    ///  
    /// - Parameter style: The style to  
    use when displaying this text.  
    /// - Returns: A text view that uses  
    the color value you supply.  
    @available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
    nonisolated public func  
foregroundStyle<S>(_ style: S) -> Text  
where S : ShapeStyle
```

```
    /// Sets the default font for text in
the view.
    ///
    /// Use `font(_:)` to apply a
specific font to an individual
    /// Text View, or all of the text
views in a container.
    ///
    /// In the example below, the first
text field has a font set directly,
    /// while the font applied to the
following container applies to all of the
    /// text views inside that container:
    ///
    ///     VStack {
    ///         Text("Font applied to a
text view.")
    ///             .font(.largeTitle)
    ///
    ///         VStack {
    ///             Text("These two text
views have the same font")
    ///                 Text("applied to
their parent view.")
    ///             }
    ///             .font(.system(size: 16,
weight: .light, design: .default))
    ///         }
    ///
    ///
    /// ! [Applying a font to a single
text view or a view container](SwiftUI-
view-font.png)
```

```
///  
/// - Parameter font: The font to use  
when displaying this text.  
/// - Returns: Text that uses the  
font you specify.  
nonisolated public func font(font:  
Font?) -> Text  
  
/// Sets the font weight of the text.  
///  
/// - Parameter weight: One of the  
available font weights.  
///  
/// - Returns: Text that uses the  
font weight you specify.  
nonisolated public func fontWeight(  
weight: Font.Weight?) -> Text  
  
/// Sets the font width of the text.  
///  
/// - Parameter width: One of the  
available font widths.  
///  
/// - Returns: Text that uses the  
font width you specify, if available.  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
nonisolated public func fontWeight(  
width: Font.Width?) -> Text  
  
/// Applies a bold font weight to the  
text.  
///
```

```
    /// - Returns: Bold text.  
    nonisolated public func bold() ->  
Text  
  
        /// Applies a bold font weight to the  
text.  
        ///  
        /// - Parameter isActive: A Boolean  
value that indicates  
        ///     whether text has bold styling.  
        ///  
        /// - Returns: Bold text.  
        @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
        nonisolated public func bold(_  
isActive: Bool) -> Text  
  
        /// Applies italics to the text.  
        ///  
        /// - Returns: Italic text.  
        nonisolated public func italic() ->  
Text  
  
        /// Applies italics to the text.  
        ///  
        /// - Parameter isActive: A Boolean  
value that indicates  
        ///     whether italic styling is  
added.  
        ///  
        /// - Returns: Italic text.  
        @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)
```

```
nonisolated public func italic(_  
isActive: Bool) -> Text  
  
    /// Modifies the font of the text to  
use the fixed-width variant  
    /// of the current font, if possible.  
    ///  
    /// – Parameter isActive: A Boolean  
value that indicates  
    /// whether monospaced styling is  
added. Default value is `true`.  
    ///  
    /// – Returns: Monospaced text.  
@available(iOS 16.4, macOS 13.3, tvOS  
16.4, watchOS 9.4, *)  
nonisolated public func monospaced(_  
isActive: Bool = true) -> Text  
  
    /// Sets the font design of the text.  
    ///  
    /// – Parameter design: One of the  
available font designs.  
    ///  
    /// – Returns: Text that uses the  
font design you specify.  
@available(iOS 16.1, macOS 13.0, tvOS  
16.1, watchOS 9.1, *)  
nonisolated public func fontDesign(_  
design: Font.Design?) -> Text  
  
    /// Modifies the text view's font to  
use fixed-width digits, while leaving  
    /// other characters proportionally
```

spaced.

```
///  
/// This modifier only affects  
numeric characters, and leaves all other  
/// characters unchanged.  
///  
/// The following example shows the  
effect of `monospacedDigit()` on a  
/// text view. It arranges two text  
views in a ``VStack``, each displaying  
/// a formatted date that contains  
many instances of the character 1.  
/// The second text view uses the  
`monospacedDigit()`. Because 1 is  
/// usually a narrow character in  
proportional fonts, applying the  
/// modifier widens all of the 1s,  
and the text view as a whole.  
/// The non-digit characters in the  
text view remain unaffected.  
///  
///     let myDate = DateComponents(  
///         calendar:  
Calendar(identifier: .gregorian),  
///         timeZone:  
TimeZone(identifier: "EST"),  
///         year: 2011,  
///         month: 1,  
///         day: 11,  
///         hour: 11,  
///         minute: 11  
///     ).date!  
///
```

```
    ///      var body: some View {
    ///
VStack(alignment: .leading) {
    ///
Text(myDate.formatted(date: .long,
time: .complete))
    ///
        .font(.system(size: 20))
    ///
Text(myDate.formatted(date: .long,
time: .complete))
    ///
        .font(.system(size: 20))
    ///
        .monospacedDigit()
}
///
///
///
/// ! [Two vertically stacked text
views, displaying the date January 11,
/// 2011, 11:11:00 AM. The second
text view uses fixed-width digits,
causing
/// all of the 1s to be wider than in
the first text
/// view.](Text-monospacedDigit-1)
///
///
/// If the base font of the text view
doesn't support fixed-width digits,
/// the font remains unchanged.
```

```
///  
/// - Returns: A text view with a  
modified font that uses fixed-width  
/// numeric characters, while leaving  
other characters proportionally  
/// spaced.  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
nonisolated public func  
monospacedDigit() -> Text
```

```
/// Applies a strikethrough to the  
text.
```

```
///  
/// - Parameters:  
/// - isActive: A Boolean value  
that indicates whether the text has a  
/// strikethrough applied.  
/// - color: The color of the  
strikethrough. If `color` is `nil`, the  
/// strikethrough uses the  
default foreground color.
```

```
///  
/// - Returns: Text with a line  
through its center.
```

```
nonisolated public func  
strikethrough(_ isActive: Bool = true,  
color: Color? = nil) -> Text
```

```
/// Applies a strikethrough to the  
text.
```

```
///  
/// - Parameters:
```

```
    /// - isActive: A Boolean value  
    // that indicates whether strikethrough  
    // is added. The default value  
    // is `true`.  
    /// - pattern: The pattern of the  
line.  
    /// - color: The color of the  
strikethrough. If `color` is `nil`, the  
    // strikethrough uses the  
default foreground color.  
    ///  
    /// - Returns: Text with a line  
through its center.  
    @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
    nonisolated public func  
strikethrough(_ isActive: Bool = true,  
pattern: Text.LineStyle.Pattern, color:  
Color? = nil) -> Text  
  
    /// Applies an underline to the text.  
    ///  
    /// - Parameters:  
    /// - isActive: A Boolean value  
that indicates whether the text has an  
    // underline.  
    /// - color: The color of the  
underline. If `color` is `nil`, the  
    // underline uses the default  
foreground color.  
    ///  
    /// - Returns: Text with a line  
running along its baseline.
```

```
nonisolated public func underline(_  
isActive: Bool = true, color: Color? =  
nil) -> Text  
  
    /// Applies an underline to the text.  
    ///  
    /// - Parameters:  
    ///     - isActive: A Boolean value  
    ///         that indicates whether underline  
    ///             styling is added. The default  
    ///             value is `true`.  
    ///     - pattern: The pattern of the  
    ///         line.  
    ///     - color: The color of the  
    ///         underline. If `color` is `nil`, the  
    ///             underline uses the default  
    ///             foreground color.  
    ///  
    /// - Returns: Text with a line  
    ///         running along its baseline.  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
nonisolated public func underline(_  
isActive: Bool = true, pattern:  
Text.LineStyle.Pattern, color: Color? =  
nil) -> Text  
  
    /// Sets the spacing, or kerning,  
    /// between characters.  
    ///  
    /// Kerning defines the offset, in  
    /// points, that a text view should shift  
    /// characters from the default
```

spacing. Use positive kerning to widen the

```
    /// spacing between characters. Use negative kerning to tighten the spacing
    /// between characters.
    ///
    ///     VStack(alignment: .leading) {
    ///
    Text("ABCDEF").kerning(-3)
        ///
        ///         Text("ABCDEF")
        ///         Text("ABCDEF").kerning(3)
        ///
        ///     }
        ///
        /// The last character in the first case, which uses negative kerning,
        /// experiences cropping because the kerning affects the trailing edge of
        /// the text view as well.
        ///
        /// ! [Three text views showing character groups, with progressively
        /// increasing spacing between the characters in each
        /// group.] (SwiftUI-Text-kerning-1.png)
        ///
        /// Kerning attempts to maintain ligatures. For example, the Hoefler Text
        /// font uses a ligature for the letter combination _ffl_, as in the word
        /// _raffle_, shown here with a small negative and a small positive kerning:
        ///
```

```
    /// ! [Two text views showing the word
raffle in the Hoefler Text font, the
    /// first with small negative and the
second with small positive kerning.
    /// The letter combination ffl has
the same shape in both variants because
    /// it acts as a ligature.] (SwiftUI-
Text-kerning-2.png)
    ///
    /// The *ffl* letter combination
keeps a constant shape as the other
letters
    /// move together or apart. Beyond a
certain point in either direction,
    /// however, kerning does disable
nonessential ligatures.
    ///
    /// ! [Two text views showing the word
raffle in the Hoefler Text font, the
    /// first with large negative and the
second with large positive kerning.
    /// The letter combination ffl does
not act as a ligature in either
    /// case.] (SwiftUI-Text-
kerning-3.png)
    ///
    /// - Important: If you add both the
``Text/tracking(_:)`` and
    /// ``Text/kerning(_:)`` modifiers
to a view, the view applies the
    /// tracking and ignores the
kerning.
    ///
```

```
    /// - Parameter kerning: The spacing  
    to use between individual characters in  
    /// this text. Value of `0` sets  
the kerning to the system default value.
```

```
    ///
```

```
    /// - Returns: Text with the  
specified amount of kerning.
```

```
nonisolated public func kerning(_  
kerning: CGFloat) -> Text
```

```
    /// Sets the tracking for the text.
```

```
    ///
```

```
    /// Tracking adds space, measured in  
points, between the characters in the  
/// text view. A positive value  
increases the spacing between characters,  
/// while a negative value brings the  
characters closer together.
```

```
    ///
```

```
    ///     VStack(alignment: .leading) {
```

```
    ///
```

```
Text("ABCDEF").tracking(-3)
```

```
    ///             Text("ABCDEF")
```

```
    ///
```

```
Text("ABCDEF").tracking(3)
```

```
    ///         }
```

```
    ///
```

```
    /// The code above uses an unusually  
large amount of tracking to make it
```

```
    /// easy to see the effect.
```

```
    ///
```

```
    /// ! [Three text views showing  
character groups with progressively
```

```
    /// increasing spacing between the
    characters in each
    /// group.] (SwiftUI-Text-
tracking.png)
    ///
    /// The effect of tracking resembles
    that of the ``Text/kerning(_:)``
    /// modifier, but adds or removes
    trailing whitespace, rather than changing
    /// character offsets. Also, using
    any nonzero amount of tracking disables
    /// nonessential ligatures, whereas
    kerning attempts to maintain ligatures.
    ///
    /// - Important: If you add both the
    ``Text/tracking(_:)`` and
    /// ``Text/kerning(_:)`` modifiers
    to a view, the view applies the
    /// tracking and ignores the
    kerning.
    ///
    /// - Parameter tracking: The amount
    of additional space, in points, that
    /// the view should add to each
    character cluster after layout. Value of
    `0`
    /// sets the tracking to the system
    default value.
    ///
    /// - Returns: Text with the
    specified amount of tracking.
    nonisolated public func tracking(_
tracking: CGFloat) -> Text
```

```
    /// Sets the vertical offset for the
    text relative to its baseline.
    ///
    /// Change the baseline offset to
    move the text in the view (in points) up
    /// or down relative to its baseline.
The bounds of the view expand to
    /// contain the moved text.
    ///
    ///     HStack(alignment: .top) {
    ///         Text("Hello")
    ///             .baselineOffset(-10)
    ///             .border(Color.red)
    ///         Text("Hello")
    ///             .border(Color.green)
    ///         Text("Hello")
    ///             .baselineOffset(10)
    ///             .border(Color.blue)
    ///     }
    ///     .background(Color(white:
0.9))
    ///
    /// By drawing a border around each
text view, you can see how the text
    /// moves, and how that affects the
view.
    ///
    /// ! [Three text views, each with the
word "Hello" outlined by a border and
    /// aligned along the top edges. The
first and last are larger than the
    /// second, with padding inside the
```

```
border above the word "Hello" in the
    /// first case, and padding inside
the border below the word in the last
    /// case.] (SwiftUI-Text-
baselineOffset.png)
    /**
     /// The first view, with a negative
offset, grows downward to handle the
     /// lowered text. The last view, with
a positive offset, grows upward. The
     /// enclosing ``HStack`` instance,
shown in gray, ensures all the text views
     /// remain aligned at their top edge,
regardless of the offset.
    /**
     /// - Parameter baselineOffset: The
amount to shift the text vertically (up
     /// or down) relative to its
baseline.
    /**
     /// - Returns: Text that's above or
below its baseline.
nonisolated public func
baselineOffset(_ baselineOffset: CGFloat)
-> Text
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Text {

    /// Defines text scales
    ///
```

```
    /// Text scale provides a way to pick
    a logical text scale
    /// relative to the base font which
    is used.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public struct Scale : Sendable,
Hashable {

    /// Defines default text scale
    ///
    /// When specified uses the
    default text scale.
    public static let `default`:
Text.Scale

    /// Defines secondary text scale
    ///
    /// When specified a uses a
    secondary text scale.
    public static let secondary:
Text.Scale

    /// 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:  
Text.Scale, b: Text.Scale) -> 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 }  
}  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Text {  
  
    /// Applies a text scale to the text.  
    ///  
    /// - Parameters:  
    ///     - scale: The text scale to  
apply.
```

```
    /// - isEnabled: If true the text
    scale is applied; otherwise text scale
    ///      is unchanged.
    /// - Returns: Text with the
    specified scale applied.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public func textScale(_ scale:
Text.Scale, isEnabled: Bool = true) ->
Text
}
```

```
extension Text {
```

```
    /// Creates a text view that displays
the current system time as defined by the
    /// given format style, keeping the
text up to date as time progresses.
    ///
    /// Use this initializer to create a
text view that updates as time
progresses, just
    /// like ``init(_:style:)``, but with
the flexibility of Foundation's
`FormatStyle`
    /// protocol.
    ///
    /// In the following example, the
first ``Text`` view presents the offset
to
    /// `startDate`, whereas the second
view displays a stopwatch counting from
    /// `startDate`. Both views are kept
```

up to date as time progresses.

///

/// Text(.currentDate,  
format: .offset(to: startDate))

/// Text(.currentDate,  
format: .stopwatch(startingAt:  
startDate))

///

/// ## Redaction for Reduced  
Luminance

///

/// When the text is displayed with  
reduced luminance and frame rate, it

/// automatically modifies the  
'format' or its output so it never shows  
outdated

/// information.

///

/// If the 'format' is known to  
SwiftUI and allows removing units or  
fields, SwiftUI

/// removes parts that change more  
frequently than the frame rate allows.

E.g. a

/// string like *\_13 minutes, 22*  
*seconds\_* would change to just '13  
minutes'.

///

/// Otherwise, SwiftUI inspects the  
'durationField', 'dateField', and  
'measurement'

/// attributes on the formatted  
output to determine which ranges need to

be

    /// redacted. If these attributes are  
not present, all digits are redacted  
using

```
    /// dashes.  
    @available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
    public init<V, F>(_ source:  
TimeDataSource<V>, format: F) where V ==  
F.FormatInput, F : DiscreteFormatStyle,  
F.FormatOutput == AttributedString
```

    /// Creates a text view that displays  
the current system time as defined by the  
    /// given format style, keeping the  
text up to date as time progresses.

```
    ///  
    /// Use this initializer to create a  
text view that updates as time  
progresses, just  
    /// like ``init(_:style:)``, but with  
the flexibility of Foundation's  
`FormatStyle`
```

```
    /// protocol.  
    ///  
    /// In the following example, the  
first ``Text`` view presents the current  
date and
```

```
    /// time, whereas the second view  
displays a soccer clock counting from  
`startDate`.
```

```
    /// Both views are kept up to date as  
time progresses.
```

```
///  
///     Text(.currentDate,  
format: .dateTime)  
///     Text(.durationOffset(to:  
startDate),  
format: .time(pattern: .minuteSecond))  
///  
/// ## Redaction for Reduced  
Luminance  
///  
/// When the text is displayed with  
reduced luminance and frame rate, it  
/// automatically modifies the  
`format` or its output so it never shows  
outdated  
/// information.  
///  
/// If the `format` is known to  
SwiftUI and allows removing units or  
fields, SwiftUI  
/// removes parts that change more  
frequently than the frame rate allows.  
E.g. a  
/// string like 13 minutes, 22  
seconds_ would change to just `13  
minutes`.  
///  
/// Otherwise, all digits in the  
formatted output are redacted using  
dashes.  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
public init<V, F>(_ source:
```

```
TimeDataSource<V>, format: F) where V ==  
F.FormatInput, F : DiscreteFormatStyle,  
F.FormatOutput == String  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Text.Storage : @unchecked  
Sendable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Text.Modifier : @unchecked  
Sendable {  
}  
  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
extension Text.DateStyle : Equatable {  
  
    /// 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:  
Text.DateStyle, b: Text.DateStyle) ->  
Bool  
}  
  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
extension Text.DateStyle : Codable {  
  
    /// Encodes this value into the given  
encoder.  
    ///  
    /// If the value fails to encode  
anything, `encoder` will encode an empty  
    /// keyed container in its place.  
    ///  
    /// This function throws an error if  
any values are invalid for the given  
    /// encoder's format.  
    ///  
    /// – Parameter encoder: The encoder  
to write data to.  
    public func encode(to encoder: any  
Encoder) throws  
  
    /// Creates a new instance by  
decoding from the given decoder.  
    ///  
    /// This initializer throws an error  
if reading from the decoder fails, or  
    /// if the data read is corrupted or  
otherwise invalid.  
    ///
```

```
    /// - Parameter decoder: The decoder  
    to read data from.  
    public init(from decoder: any  
Decoder) throws  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Text.TruncationMode : Equatable  
{  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Text.TruncationMode : Hashable  
{  
}  
  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
extension Text.Case : Equatable {  
}  
  
@available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
extension Text.Case : Hashable {  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Text.Layout {  
    /// Option flags used when drawing
```

```
`Text.Layout` lines or runs into
    /// a graphics context.
    @frozen public struct
DrawingOptions : OptionSet {

    /// The corresponding value of
the raw type.
    ///
    /// A new instance initialized
with `rawValue` will be equivalent to
this
    /// instance. For example:
    ///
    ///     enum PaperSize: String {
    ///         case A4, A5, Letter,
Legal
    ///     }
    ///
    ///     let selectedSize =
PaperSize.Letter
    ///
print(selectedSize.rawValue)
    ///     // Prints "Letter"
    ///
    ///     print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    ///     // Prints "true"
public let rawValue: UInt32

    /// Creates a new option set from
the given raw value.
    ///
```

```
    /// This initializer always
    succeeds, even if the value passed as
    `rawValue`  

        /// exceeds the static properties
    declared as part of the option set. This
        /// example creates an instance
    of `ShippingOptions` with a raw value
    beyond  

        /// the highest element, with a
    bit mask that effectively contains all
    the  

        /// declared static members.  

        ///  

        ///     let extraOptions =
    ShippingOptions(rawValue: 255)
        ///
print(extraOptions.isStrictSuperset(of: .
all))
        ///     // Prints "true"
        ///
        /// - Parameter rawValue: The raw
    value of the option set to create. Each
    bit  

        ///     of `rawValue` potentially
    represents an element of the option set,
        ///     though raw values may
    include bits that are not defined as
    distinct  

        ///     values of the `OptionSet`
    type.  

public init(rawValue: UInt32)  

  

        /// If set, subpixel quantization
```

```
requested by the text engine
    /// is disabled. This can be
useful for text that will be
    /// animated to prevent it
jittering.
    public static var
disablesSubpixelQuantization:
Text.Layout.DrawingOptions { get }

    /// The type of the elements of
an array literal.
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias
ArrayLiteralElement =
Text.Layout.DrawingOptions

    /// The element type of the
option set.
    /**
     /// To inherit all the default
implementations from the `OptionSet`
protocol,
    /// the `Element` type must be
`Self`, the default.
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
    public typealias Element =
Text.Layout.DrawingOptions

    /// The raw type that can be used
to represent all values of the conforming
    /// type.
```

```
    /**
     * Every distinct value of the conforming type has a corresponding unique
     * value of the `RawValue` type, but there may be values of the `RawValue`
     * type that don't have a corresponding value of the conforming type.
     */
    @available(iOS 17.0, tvOS 17.0,
watchOS 10.0, macOS 14.0, *)
public typealias RawValue =
UInt32
}

@available(iOS 17.0, macOS 14.0, tvOS 17.0, watchOS 10.0, *)
extension Text.Layout.CharacterIndex : BitwiseCopyable {

@available(iOS 17.0, macOS 14.0, tvOS 17.0, watchOS 10.0, *)
extension Text.Layout.TypographicBounds : BitwiseCopyable {

@available(iOS 17.0, macOS 14.0, tvOS 17.0, watchOS 10.0, *)
extension Text.Layout.DrawingOptions : Sendable {


```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Text.Layout.DrawingOptions :  
BitwiseCopyable {  
}  
  
/// An alignment position for text along  
the horizontal axis.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public enum TextAlignement :  
Hashable, CaseIterable {  
  
    case leading  
  
    case center  
  
    case trailing  
  
    /// 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:
```

```
    public func hash(into hasher: inout Hasher)
```

```
        /// 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)
```

```
        /// A type that can represent a
collection of all values of this type.
@available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
```

```
public typealias AllCases =
[TextAlignment]

    /// A collection of all values of
this type.
nonisolated public static var
allCases: [TextAlignment] { get }

    /// 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 }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension TextAlignment : Sendable {
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
```

```
extension TextAlignment : BitwiseCopyable
{
}

/// A value that you can attach to text
views and that text renderers can query.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public protocol TextAttribute : Hashable
{

}

/// A proxy for a text view that custom
text renderers use.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct TextProxy {

    /// Returns the space needed by the
    text view, for a proposed size.
    ///
    /// - Parameter proposal: the
    proposed size of the text view.
    ///
    /// - Returns: the size that the text
    view requires for the
    /// given proposal.
    public func sizeThatFits(_ proposal:
    ProposedViewSize) -> CGSize
}

/// A value that can replace the default
text view rendering behavior.
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
public protocol TextRenderer : Animatable  
{  
  
    /// Draws `layout` into `ctx`.  
    func draw(layout: Text.Layout, in  
    ctx: inout GraphicsContext)  
  
    /// Returns the size of the text in  
    `proposal`. The provided `text`  
    /// proxy value may be used to query  
    the sizing behavior of the  
    /// underlying text layout.  
    ///  
    /// The default implementation of  
    this function returns  
    /// `text.size(proposal)`.  
    @available(iOS 18.0, macOS 15.0, tvOS  
    18.0, watchOS 11.0, visionOS 2.0, *)  
    func sizeThatFits(proposal:  
    ProposedViewSize, text: TextProxy) ->  
    CGSize  
  
    /// Returns the size of the extra  
    padding added to any drawing  
    /// layer used to rasterize the text.  
    For example when drawing the  
    /// text with a shadow this may be  
    used to extend the drawing  
    /// bounds to avoid clipping the  
    shadow.  
    ///
```

```
    /// The default implementation of
    this function returns an empty
    /// set of insets.
    @available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
    var displayPadding: EdgeInsets {
get }
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension TextRenderer {

    /// Returns the size of the text in
    `proposal`. The provided `text`
    /// proxy value may be used to query
    the sizing behavior of the
    /// underlying text layout.
    ///
    /// The default implementation of
    this function returns
    /// `text.size(proposal)`.

    public func sizeThatFits(proposal:
ProposedViewSize, text: TextProxy) ->
CGSize

    /// Returns the size of the extra
    padding added to any drawing
    /// layer used to rasterize the text.
    For example when drawing the
    /// text with a shadow this may be
    used to extend the drawing
    /// bounds to avoid clipping the
```

shadow.

```
///  
/// The default implementation of  
this function returns an empty  
/// set of insets.  
public var displayPadding: EdgeInsets  
{ get }  
}
```

/// A type that describes the ability to  
select text.

```
///  
/// To configure whether people can  
select text in your app, use the  
/// ``View/textSelection(_:)`` modifier,  
passing in a text selectability  
/// value like ``enabled`` or  
``disabled``.
```

```
@available(iOS 15.0, macOS 12.0, *)
```

```
@available(tvOS, unavailable)
```

```
@available(watchOS, unavailable)
```

```
public protocol TextSelectability {
```

/// A Boolean value that indicates  
whether the selectability type allows

/// selection.

```
///
```

/// Conforming types, such as  
``EnabledTextSelectability`` and  
``DisabledTextSelectability``,

return `true` or `false` for this  
/// property as appropriate. SwiftUI

expects this value for a given

```
    /// selectability type to be
    constant, unaffected by global state.
    static var allowsSelection: Bool {
get }
}

@available(iOS 15.0, macOS 12.0, *)
@available(tvOS, unavailable)
@available(watchOS, unavailable)
extension TextSelectability where Self == EnabledTextSelectability {

    /// A selectability value that
    enables text selection by a person using
    your app.
    /**
     /// Enabling text selection allows
     people to perform actions on the text
     /// content, such as copying and
     sharing. Enable text selection in views
     /// where those operations are
     useful, such as copying unique IDs or
     /// error messages. This allows
     people to paste the data into
     /// emails or documents.
     /**
     /// The following example enables
     text selection on the second of two
     /// ``Text`` views in a ``VStack``.
     /**
     ///      VStack {
     ///          Text("Event Invite")
     ///              .font(.title)
```

```
    /**
     Text(invite.date.formatted(date: .long,
     time: .shortened))
        /**
         .textSelection(.enabled)
     ed)
    /**
     */
    /**
     public static var enabled:
EnabledTextSelectability { get }
}

@available(iOS 15.0, macOS 12.0, *)
@available(tvOS, unavailable)
@available(watchOS, unavailable)
extension TextSelectability where Self == DisabledTextSelectability {

    /**
     A selectability value that
disables text selection by the person
using your app.
    /**
    /**
     Use this property to disable text
selection of views that
    /**
     you don't want people to select
and copy, even if contained within an
    /**
     overall context that allows text
selection.
    /**
    /**
     content // Content that might
contain Text views.
    /**
     .textSelection(.disabled)
    /**
     .padding()
    /**
     .contentShape(Rectangle())
```

```
    /// .gesture(someGesture)
    ///
    public static var disabled: DisabledTextSelectability { get }
}

/// A protocol for controlling the size
variant of text views.
@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public protocol TextVariantPreference {

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension TextVariantPreference where
Self == FixedTextVariant {

    /// The default text variant
    preference. It always chooses the largest
    available
    /// variant.
    public static var fixed: FixedTextVariant { get }
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension TextVariantPreference where
Self == SizeDependentTextVariant {

    /// The size dependent preference
    allows the text to take the available
```

```
space into
    /// account when choosing the size
variant to display.
    /**
     /// When a ``Text`` provides
different size options for its content,
the size
    /// dependent preference chooses the
largest option that fits into the
available
    /// space without truncating or
clipping its content.
    /**
     /// - Note: Only use this option
where needed as it incurs a performance
cost on
    /// every ``Text`` it is applied to,
even if the concrete text initializer
cannot
    /// provide multiple size variants
and there is no visual impact.
    /**
     /// ## Difference to ``ViewThatFits``
    /**
     /// The ``sizeDependent`` text
variant preference differs from
``ViewThatFits`` both
    /// in usage and in behavior.
``ViewThatFits`` chooses the first child
where the
    /// **ideal** size fits the available
space. For ``Text`` this means that it
will
```

```
    /// only choose texts that can fit  
their contents into the available space  
**without
```

```
    /// a line break**. With this text  
variant preference, on the other hand,  
the
```

```
    /// largest variant is chosen that  
can fit the available space while  
respecting all
```

```
    /// the regular layout rules, such as  
``EnvironmentValues/lineLimit``.
```

```
    ///
```

```
    /// To use ``ViewThatFits``, multiple  
different views have to be provided as  
the
```

```
    /// different size options. With this  
text variant preference, a single  
``Text``
```

```
    /// provides the different size  
variants intrinsically. The way it  
generates these
```

```
    /// size variants and how many size  
variants are available depends on the  
text
```

```
    /// initializer used.
```

```
    public static var sizeDependent:  
SizeDependentTextVariant { get }  
}
```

```
/// A source of time related data.
```

```
///
```

```
/// Instances of this type provide  
``Text`` with live and automatically
```

```
updating
/// values in Widgets, Live Activities,
watchOS Complications, and of course
regular
/// apps.

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
public struct TimeDataSource<Value> {

}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension TimeDataSource : Sendable where
Value : Sendable {
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension TimeDataSource {

    /// A time data source that produces
`Date.now`.

    public static var currentDate:
TimeDataSource<Date> { get }

    /// A time data source that produces
the offset between `Date.now` and the
given
    /// `date` as a `Duration`.
    public static func durationOffset(to
date: Date) -> TimeDataSource<Duration>

    /// A time data source that produces
```

```
`date..<max(date, Date.now)`.  
    public static func  
dateRange(startingAt date: Date) ->  
TimeDataSource<Range<Date>>  
  
    /// A time data source that produces  
`min(date, Date.now)..<date`.  
    public static func dateRange(endingAt  
date: Date) ->  
TimeDataSource<Range<Date>>  
}  
  
/// A type that provides a sequence of  
dates for use as a schedule.  
///  
/// Types that conform to this protocol  
implement a particular kind of schedule  
/// by defining an  
``TimelineSchedule/entries(from:mode:)``  
method that returns  
/// a sequence of dates. Use a timeline  
schedule type when you initialize  
/// a ``TimelineView``. For example, you  
can create a timeline view that  
/// updates every second, starting from  
some `startDate`, using a  
/// periodic schedule returned by  
``TimelineSchedule/periodic(from:by:)``:  
///  
///     TimelineView(.periodic(from:  
startDate, by: 1.0)) { context in  
///         // View content goes here.  
///     }
```

```
///  
/// You can also create custom timeline  
schedules.  
/// The timeline view updates its content  
according to the  
/// sequence of dates produced by the  
schedule.  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
public protocol TimelineSchedule {  
  
    /// An alias for the timeline  
schedule update mode.  
    typealias Mode = TimelineScheduleMode  
  
    /// The sequence of dates within a  
schedule.  
    ///  
    /// The  
``TimelineSchedule/entries(from:mode:)``  
method returns a value  
    /// of this type, which is a  
    ///  
<doc://com.apple.documentation/documentation/Swift/Sequence>  
    /// of dates in ascending order. A  
``TimelineView`` that you create with a  
    /// schedule updates its content at  
the moments in time corresponding to  
    /// the dates included in the  
sequence.  
    associatedtype Entries : Sequence  
where Self.Entries.Element == Date
```

```
    /// Provides a sequence of dates
    starting around a given date.
    ///
    /// A ``TimelineView`` that you
    create calls this method to figure out
    /// when to update its content. The
    method returns a sequence of dates in
    /// increasing order that represent
    points in time when the timeline view
    /// should update. Types that conform
    to the ``TimelineSchedule`` protocol,
    /// like the one returned by
    ``TimelineSchedule/periodic(from:by:)``,
    or a custom schedule that
    /// you define, implement a custom
    version of this method to implement a
    /// particular kind of schedule.
    ///
    /// One or more dates in the sequence
    might be before the given
    /// `startDate`, in which case the
    timeline view performs its first
    /// update at `startDate` using the
    entry that most closely precedes
    /// that date. For example, if in
    response to a `startDate` of
    /// `10:09:55`, the method returns a
    sequence with the values `10:09:00`,
    /// `10:10:00`, `10:11:00`, and so
    on, the timeline view performs an initial
    /// update at `10:09:55` (using the
    `10:09:00` entry), followed by another
```

```
    /// update at the beginning of every
    minute, starting at `10:10:00`.
    /**
     * A type that conforms should
     * adjust its behavior based on the `mode`
     * when
     *   possible. For example, a periodic
     * schedule providing updates
     *   for a timer could restrict
     * updates to once per minute while in the
     */
``TimelineScheduleMode/lowFrequency``
mode:
    /**
     *   func entries(
     *       from startDate: Date,
mode: TimelineScheduleMode
     *   ) -> PeriodicTimelineSchedule
{
    /**
     *       .periodic(
     *           from: startDate, by:
(mode == .lowFrequency ? 60.0 : 1.0)
     *               )
     *   }
     /**
     * - Parameters:
     *   - startDate: The date by which
the sequence begins.
     *   - mode: An indication of
whether the schedule updates normally,
     *   or with some other cadence.
     * - Returns: A sequence of dates in
ascending order.
```

```
func entries(from startDate: Date,  
mode: Self.Mode) -> Self.Entries  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension TimelineSchedule where Self ==  
PeriodicTimelineSchedule {  
  
    /// A schedule for updating a  
    timeline view at regular intervals.  
    ///  
    /// Initialize a ``TimelineView``  
    with a periodic timeline schedule when  
    /// you want to schedule timeline  
    view updates periodically with a custom  
    /// interval:  
    ///  
    ///     TimelineView(.periodic(from:  
    startDate, by: 3.0)) { context in  
    ///  
    Text(context.date.description)  
    ///      }  
    ///  
    /// The timeline view updates its  
    content at the start date, and then  
    /// again at dates separated in time  
    by the interval amount, which is every  
    /// three seconds in the example  
    above. For a start date in the  
    /// past, the view updates  
    immediately, providing as context the  
    date
```

```
    /// corresponding to the most recent
interval boundary. The view then
    /// refreshes normally at subsequent
interval boundaries. For a start date
    /// in the future, the view updates
once with the current date, and then
    /// begins regular updates at the
start date.
    ///
    /// The schedule defines the
``PeriodicTimelineSchedule/Entries``
    /// structure to return the sequence
of dates when the timeline view calls
    /// the
``PeriodicTimelineSchedule/entries(from:m
ode:)`` method.
    ///
    /// - Parameters:
    ///   - startDate: The date on which
to start the sequence.
    ///   - interval: The time interval
between successive sequence entries.
    public static func periodic(from
startDate: Date, by interval:
TimeInterval) -> PeriodicTimelineSchedule
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension TimelineSchedule where Self ==
EveryMinuteTimelineSchedule {

    /// A schedule for updating a
```

timeline view at the start of every minute.

```
///
/// Initialize a ``TimelineView``
with an every minute timeline schedule
    /// when you want to schedule
timeline view updates at the start of
every
    /// minute:
    ///
    ///     TimelineView(.everyMinute)
{ context in
    ///
Text(context.date.description)
    /// }
```

///
 /// The schedule provides the first date as the beginning of the minute in
 /// which you use it to initialize the timeline view. For example, if you
 /// create the timeline view at `10:09:38`, the schedule's first entry is
 /// `10:09:00`. In response, the timeline view performs its first update
 /// immediately, providing the beginning of the current minute, namely
 /// `10:09:00`, as context to its content. Subsequent updates happen at the
 /// beginning of each minute that follows.

///
 /// The schedule defines the ``EveryMinuteTimelineSchedule/Entries``

```
    /// structure to return the sequence
    of dates when the timeline view calls
    /// the
``EveryMinuteTimelineSchedule/entries(fro
m:mode:)`` method.

    public static var everyMinute:
EveryMinuteTimelineSchedule { get }
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension TimelineSchedule {

    /// A schedule for updating a
    timeline view at explicit points in time.
    ///
    /// Initialize a ``TimelineView``
    with an explicit timeline schedule when
    /// you want to schedule view updates
    at particular points in time:
    ///
    ///     let dates = [
    ///
Date(timeIntervalSinceNow: 10), // Update
ten seconds from now,
    ///
Date(timeIntervalSinceNow: 12) // and a
few seconds later.
    ///
    ]
    ///
    ///     struct MyView: View {
    ///         var body: some View {
    ///
```

```
TimelineView(.explicit(dates)) { context
in
    /**
     * - Parameter dates: The sequence
     * of dates at which a timeline view
     * updates. Use a monotonically
     * increasing sequence of dates,
     * and ensure that at least one is
     * in the future.
     */
    public static func explicit<S>(_
dates: S) -> ExplicitTimelineSchedule<S>
```

```
where Self ==  
ExplicitTimelineSchedule<S>, S :  
Sequence, S.Element == Date  
}  
  
/// A mode of operation for timeline  
schedule updates.  
///  
/// A ``TimelineView`` provides a mode  
when calling its schedule's  
///  
``TimelineSchedule/entries(from:mode:)``  
method.  
/// The view chooses a mode based on the  
state of the system.  
/// For example, a watchOS view might  
request a lower frequency  
/// of updates, using the  
``lowFrequency`` mode, when the user  
/// lowers their wrist.  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
public enum TimelineScheduleMode :  
Sendable {  
  
    /// A mode that produces schedule  
updates at the schedule's natural  
cadence.  
    case normal  
  
    /// A mode that produces schedule  
updates at a reduced rate.  
    ///
```

```
    /// In this mode, the schedule should
    generate only
    /// "major" updates, if possible. For
    example, a timeline providing
    /// updates to a timer might restrict
    updates to once a minute while in
    /// this mode.
    case lowFrequency

        /// 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:
TimelineScheduleMode, b:
TimelineScheduleMode) -> Bool

        /// 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)
```

```
    /// 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 }  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension TimelineScheduleMode :  
Equatable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension TimelineScheduleMode : Hashable  
{  
}  
  
/// A style that reflects the current  
tint color.  
///  
/// You can set the tint color with the  
``View/tint(_:)`` modifier. If no  
/// explicit tint is set, the tint is  
derived from the app's accent color.  
///  
/// You can also use ``ShapeStyle/tint``  
to construct this style.  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
public struct TintShapeStyle : ShapeStyle  
{  
  
    /// Creates a tint shape style.  
    public init()
```

```
    /// The type of shape style this will
    resolve to.
    ///
    /// When you create a custom shape
    style, Swift infers this type
    /// from your implementation of the
    required `resolve` function.
    @available(iOS 17.0, tvOS 17.0,
    watchOS 10.0, macOS 14.0, *)
    public typealias Resolved = Never
}

/// The context of the current state-
processing update.
///
/// Use a transaction to pass an
animation between views in a view
hierarchy.
///
/// The root transaction for a state
change comes from the binding that
changed,
/// plus any global values set by calling
``withTransaction(_:_:)`` or
/// ``withAnimation(_:_:)``.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct Transaction {

    /// Creates a transaction.
    @inlinable public init()
```

```
    /// Accesses the transaction value
    /// associated with a custom key.
    ///
    /// Create custom transaction values
    by defining a key that conforms to the
    /// ``TransactionKey`` protocol, and
    then using that key with the subscript
    /// operator of the ``Transaction``
    structure to get and set a value for
    /// that key:
    ///
    ///     private struct
MyTransactionKey: TransactionKey {
    ///             static let defaultValue =
false
    ///
    ///
    ///     extension Transaction {
    ///             var myCustomValue: Bool {
    ///                 get
{ self[MyTransactionKey.self] }
    ///                 set
{ self[MyTransactionKey.self] =
newValue }
    ///
    ///
    /// @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    public subscript<K>(key: K.Type) ->
K.Value where K : TransactionKey
}

@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
extension Transaction {

    /// A Boolean value that indicates
    whether the transaction originated from
    /// an action that produces a
    sequence of values.
    ///
    /// This value is `true` if a
    continuous action created the
    transaction, and
    /// is `false` otherwise. Continuous
    actions include things like dragging a
    /// slider or pressing and holding a
    stepper, as opposed to tapping a
    /// button.
    public var isContinuous: Bool
}

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

    /// The preferred alignment of the
    view within a scroll view's visible
    /// region when scrolling to a view.
    ///
    /// Use this API in conjunction with
    a
    ///
    ``ScrollViewProxy/scrollTo(_:anchor)`` or
    when updating the binding
    /// provided to a
```

```
``View/scrollPosition(id:anchor:)``.  
    ///  
    ///      @Binding var position:  
Item.ID?  
    ///  
    ///      var body: some View {  
    ///          ScrollView {  
    ///              LazyVStack {  
    ///                  ForEach(items)  
{ item in  
    ///      ItemView(item)  
    ///      }  
    ///      }  
    ///      .scrollTargetLayout()  
}  
    ///      .scrollPosition(id:  
$position)  
    ///      .safeAreaInset(edge: .bot  
tom) {  
    ///          Button("Scroll To  
Bottom") {  
    ///              withAnimation {  
withTransaction(\.scrollTargetAnchor, .bo  
ttom) {  
    ///                  position  
= items.last?.id  
    ///              }  
    ///          }  
    ///      }  
    ///  }  
    /// }
```

```
    /**
     * When used with the
     ``View/scrollPosition(id:anchor:)``
     modifier,
        /// this value will be preferred over
     the anchor specified in the
        /// modifier for the current
     transaction.
        @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
        public var scrollTargetAnchor:
UnitPoint?
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension Transaction {

    /// The behavior a scroll view will
    have regarding content offset
    /// adjustments for the current
    transaction.
    ///
    /// A scroll view may automatically
    adjust its content offset
    /// based on the current context. The
    absolute offset may be adjusted
    /// to keep content in relatively the
    same place. For example,
    /// when scrolled to the bottom, a
    scroll view may keep the bottom
    /// edge scrolled to the bottom when
    the overall size of its content
```

```
    /// changes.  
    ///  
    /// Use this property to disable  
    these kinds of adjustments when needed.  
    @available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
    public var  
scrollContentOffsetAdjustmentBehavior:  
    ScrollContentOffsetAdjustmentBehavior  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Transaction {  
  
    /// Creates a transaction and assigns  
    its animation property.  
    ///  
    /// - Parameter animation: The  
    animation to perform when the current  
    state  
    /// changes.  
    public init(animation: Animation?)  
  
    /// The animation, if any, associated  
    with the current state change.  
    public var animation: Animation?  
  
    /// A Boolean value that indicates  
    whether views should disable animations.  
    ///  
    /// This value is `true` during the  
    initial phase of a two-part transition
```

```
    /// update, to prevent
``View/animation(_:)`` from inserting new
animations
    /// into the transaction.
    public var disablesAnimations: Bool
}

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

    /// Whether this transaction will
track the velocity of any animatable
    /// properties that change.
    ///
    /// This property can be enabled in
an interactive context to track velocity
    /// during a user interaction so that
when the interaction ends, an
    /// animation can use the accumulated
velocities to create animations that
    /// preserve them. This tracking is
mutually exclusive with an animation
    /// being used during a view change,
since if there is an animation, it is
    /// responsible for managing its own
velocity.
    ///
    /// Gesture onChanged and updating
callbacks automatically set this property
    /// to true.
    ///
    /// This example shows an interaction
```

```
which applies changes, tracking
    /// velocity until the final change,
which applies an animation (which will
    /// start with the velocity that was
tracked during the previous changes).
    /// These changes could come from a
server or from an interactive control
    /// like a slider.
    ///
    ///     func receiveChange(change:
ChangeInfo) {
        ///         var transaction =
Transaction()
        ///
        ///         if change.isFinal {
            transaction.animation
= .spring
        ///             } else {
        ///
transaction.tracksVelocity = true
        ///
        ///         }
        ///
withTransaction(transaction) {
    ///
state.applyChange(change)
    ///
    ///         }
    ///
    ///     }
public var tracksVelocity: Bool
}

/// A key for accessing values in a
transaction.
///
/// You can create custom transaction
```

```
values by extending the ``Transaction``  
/// structure with new properties.  
/// First declare a new transaction key  
type and specify a value for the  
/// required ``defaultValue`` property:  
///  
///     private struct MyTransactionKey:  
TransactionKey {  
///         static let defaultValue =  
false  
///     }  
///  
/// The Swift compiler automatically  
infers the associated ``Value`` type as  
the  
/// type you specify for the default  
value. Then use the key to define a new  
/// transaction value property:  
///  
///     extension Transaction {  
///         var myCustomValue: Bool {  
///             get  
{ self[MyTransactionKey.self] }  
///             set  
{ self[MyTransactionKey.self] =  
newValue }  
///         }  
///     }  
///  
/// Clients of your transaction value  
never use the key directly.  
/// Instead, they use the key path of  
your custom transaction value property.
```

```
/// To set the transaction value for a
change, wrap that change in a call to
/// `withTransaction`:
///
///     withTransaction(\.myCustomValue,
true) {
///         isActive.toggle()
///     }
///
/// To use the value from inside `MyView`
or one of its descendants, use the
/// ``View/transaction(_:)`` view
modifier:
///
///     MyView()
///         .transaction { transaction in
///             if
transaction.myCustomValue {
///                 transaction.animation
= .default.repeatCount(3)
///             }
///         }
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public protocol TransactionKey {

    /// The associated type representing
the type of the transaction key's
    /// value.
    associatedtype Value

    /// The default value for the
transaction key.
```

```
    static var defaultValue: Self.Value {  
get }  
}  
  
/// A shape with an affine transform  
// applied to it.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
@frozen public struct  
TransformedShape<Content> : Shape where  
Content : Shape {  
  
    public var shape: Content  
  
    public var transform:  
CGAffineTransform  
  
    @inlinable public init(shape:  
Content, transform: CGAffineTransform)  
  
        /// Describes this shape as a path  
within a rectangular frame of reference.  
        ///  
        /// - Parameter rect: The frame of  
reference for describing this shape.  
        ///  
        /// - Returns: A path that describes  
this shape.  
    nonisolated public func path(in rect:  
CGRect) -> Path  
  
        /// An indication of how to style a  
shape.
```

```
///  
/// SwiftUI looks at a shape's role  
when deciding how to apply a  
/// ``ShapeStyle`` at render time.  
The ``Shape`` protocol provides a  
/// default implementation with a  
value of ``ShapeRole/fill``. If you  
/// create a composite shape, you can  
provide an override of this property  
/// to return another value, if  
appropriate.  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
nonisolated public static var role:  
ShapeRole { get }  
  
/// Returns the behavior this shape  
should use for different layout  
/// directions.  
///  
/// If the layoutDirectionBehavior  
for a Shape is one that mirrors, the  
/// shape's path will be mirrored  
horizontally when in the specified layout  
/// direction. When mirrored, the  
individual points of the path will be  
/// transformed.  
///  
/// Defaults to `mirrors` when  
deploying on iOS 17.0, macOS 14.0,  
/// tvOS 17.0, watchOS 10.0 and  
later, and to `fixed` if not.  
/// To mirror a path when deploying
```

```
to earlier releases, either use
    /**
`View.flipsForRightToLeftLayoutDirection`
for a filled or stroked
    /// shape or conditionally mirror the
points in the path of the shape.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    nonisolated public var
layoutDirectionBehavior:
LayoutDirectionBehavior { get }

    /// The type defining the data to
animate.
    public typealias AnimatableData =
Content.AnimatableData

    /// The data to animate.
    public var animatableData:
TransformedShape<Content>.AnimatableData

    /// The type of view representing the
body of this view.
    /**
    /// When you create a custom view,
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Body
}
```

```
/// A description of view changes to
/// apply when a view is added to and removed
/// from the view hierarchy.
///
/// A transition should generally be made
by applying one or more modifiers to
/// the `content`. For symmetric
transitions, the `isIdentity` property on
/// `phase` can be used to change the
properties of modifiers. For asymmetric
/// transitions, the phase itself can be
used to change those properties.
/// Transitions should not use any
identity-affecting changes like ` `.id` ,
`if` ,
/// and `switch` on the `content`, since
doing so would reset the state of the
/// view they're applied to, causing
wasted work and potentially surprising
/// behavior when it appears and
disappears.
///
/// The following code defines a
transition that can be used to change the
/// opacity and rotation when a view
appears and disappears.
///
///     struct RotatingFadeTransition:
Transition {
///         func body(content: Content,
phase: TransitionPhase) -> some View {
///             content
///                 .opacity(phase.isIdent
```

```
ty ? 1.0 : 0.0)
///                                .rotationEffect(phase.r
otation)
///                                }
///                                }
/// extension TransitionPhase {
///     fileprivate var rotation:
Angle {
///             switch self {
///             case .willAppear:
return .degrees(30)
///             case .identity:
return .zero
///             case .didDisappear:
return .degrees(-30)
///             }
///             }
///             }
/// - See Also: `TransitionPhase`
/// - See Also: `AnyTransition`
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@MainActor @preconcurrency public
protocol Transition {

    /// The type of view representing the
body.
    associatedtype Body : View

    /// Gets the current body of the
caller.
    ///
```

```
    /// `content` is a proxy for the view
    // that will have the modifier
    /// represented by `Self` applied to
    // it.
    @ViewBuilder @MainActor
    @preconcurrency func body(content:
        Self.Content, phase: TransitionPhase) ->
        Self.Body

    /// Returns the properties this
    // transition type has.
    ///
    /// Defaults to
    `TransitionProperties()`.

    @MainActor @preconcurrency static var
    properties: TransitionProperties { get }

    /// The content view type passed to
    `body()`.

    typealias Content =
    PlaceholderContentView<Self>
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Transition where Self ==
OffsetTransition {

    /// Returns a transition that offset
    // the view by the specified amount.
    @MainActor @preconcurrency public
    static func offset(_ offset: CGSize) ->
    Self
```

```
    /// Returns a transition that offset  
the view by the specified x and y  
    /// values.  
    @MainActor @preconcurrency public  
static func offset(x: CGFloat = 0, y:  
CGFloat = 0) -> Self  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Transition where Self ==  
MoveTransition {  
  
    /// Returns a transition that moves  
the view away, towards the specified  
    /// edge of the view.  
    @MainActor @preconcurrency public  
static func move(edge: Edge) -> Self  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Transition where Self ==  
OpacityTransition {  
  
    /// A transition from transparent to  
opaque on insertion, and from opaque to  
    /// transparent on removal.  
    @MainActor @preconcurrency public  
static var opacity: OpacityTransition {  
get }  
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Transition where Self ==  
SlideTransition {  
  
    /// A transition that inserts by  
    moving in from the leading edge, and  
    /// removes by moving out towards the  
    trailing edge.  
    ///  
    /// - SeeAlso:  
    `AnyTransition.move(edge:)`  
    @MainActor @preconcurrency public  
    static var slide: SlideTransition { get }  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Transition where Self ==  
PushTransition {  
  
    /// Creates a transition that when  
    added to a view will animate the  
    /// view's insertion by moving it in  
    from the specified edge while  
    /// fading it in, and animate its  
    removal by moving it out towards  
    /// the opposite edge and fading it  
    out.  
    ///  
    /// - Parameters:  
    ///     - edge: the edge from which the
```

```
view will be animated in.  
    ///  
    /// - Returns: A transition that  
    animates a view by moving and  
    /// fading it.  
    @MainActor @preconcurrency public  
    static func push(from edge: Edge) -> Self  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Transition {  
  
    /// Attaches an animation to this  
    transition.  
    @MainActor @preconcurrency public  
    func animation(_ animation: Animation?)  
-> some Transition  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Transition {  
  
    @MainActor @preconcurrency public  
    func combined<T>(with other: T) -> some  
    Transition where T : Transition  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)
```

```
extension Transition {

    /// Returns the properties this
    transition type has.
    ///
    /// Defaults to
    `TransitionProperties()`.

    @MainActor @preconcurrency public
    static var properties:
    TransitionProperties { get }

    @MainActor @preconcurrency public
    func apply<V>(content: V, phase:
    TransitionPhase) -> some View where V :
    View

}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Transition where Self ==
IdentityTransition {

    /// A transition that returns the
    input view, unmodified, as the output
    /// view.
    @MainActor @preconcurrency public
    static var identity: IdentityTransition {
    get }
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
```

```
extension Transition where Self ==  
BlurReplaceTransition {  
  
    /// A transition that animates the  
    insertion or removal of a view  
    /// by combining blurring and scaling  
    effects.  
    @MainActor @preconcurrency public  
    static func blurReplace(_ config:  
    BlurReplaceTransition.Configuration  
    = .downUp) -> Self  
  
    /// A transition that animates the  
    insertion or removal of a view  
    /// by combining blurring and scaling  
    effects.  
    @MainActor @preconcurrency public  
    static var blurReplace:  
    BlurReplaceTransition { get }  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Transition where Self ==  
ScaleTransition {  
  
    /// Returns a transition that scales  
    the view.  
    @MainActor @preconcurrency public  
    static var scale: ScaleTransition { get }  
  
    /// Returns a transition that scales  
    the view by the specified amount.
```

```
    @MainActor @preconcurrency public
    static func scale(_ scale: Double,
                      anchor: UnitPoint = .center) -> Self
}

/// An indication of which the current
stage of a transition.
///
/// When a view is appearing with a
transition, the transition will first be
/// shown with the `willAppear` phase,
then will be immediately moved to the
/// `identity` phase. When a view is
being removed, its transition is changed
/// from the `identity` phase to the
`didDisappear` phase. If a view is
removed
/// while it is still transitioning in,
then its phase will change to
/// `didDisappear`. If a view is re-added
while it is transitioning out, its
/// phase will change back to `identity`.
///
/// In the `identity` phase, transitions
should generally not make any visual
/// change to the view they are applied
to, since the transition's view
/// modifications in the `identity` phase
will be applied to the view as long as
/// it is visible. In the `willAppear`
and `didDisappear` phases, transitions
/// should apply a change that will be
animated to create the transition. If no
```

```
/// animatable change is applied, then
the transition will be a no-op.
///
/// - See Also: `Transition`
/// - See Also: `AnyTransition`
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
@frozen public enum TransitionPhase {

    /// The transition is being applied
    to a view that is about to be inserted
    /// into the view hierarchy.
    ///
    /// In this phase, a transition
    should show the appearance that will be
    /// animated from to make the
    appearance transition.

    case willAppear

    /// The transition is being applied
    to a view that is in the view hierarchy.
    ///
    /// In this phase, a transition
    should show its steady state appearance,
    /// which will generally not make any
    visual change to the view.

    case identity

    /// The transition is being applied
    to a view that has been requested to be
    /// removed from the view hierarchy.
    ///
    /// In this phase, a transition
```

```
should show the appearance that will be
    /// animated to to make the
disappearance transition.
case didDisappear

    /// A Boolean that indicates whether
the transition should have an identity
    /// effect, i.e. not change the
appearance of its view.
    ///
    /// This is true in the `identity`
phase.
public var isIdentity: Bool { get }

    /// 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:
TransitionPhase, b: TransitionPhase) ->
Bool

    /// 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)  
  
/// 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 }  
}
```

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

```
    /// A value that can be used to  
multiply effects that are applied
```

```
    /// differently depending on the  
phase.
```

```
    ///  
    /// - Returns: Zero when in the  
`identity` case, -1.0 for `willAppear`,  
    /// and 1.0 for `didDisappear`.  
public var value: Double { get }  
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension TransitionPhase : Equatable {  
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension TransitionPhase : Hashable {  
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension TransitionPhase : Sendable {  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension TransitionPhase :  
BitwiseCopyable {  
}  
  
/// The properties a `Transition` can  
have.  
///  
/// A transition can have properties that  
specify high level information about  
/// it. This can determine how a  
transition interacts with other features  
like  
/// Accessibility settings.  
///  
/// - See Also: `Transition`  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
public struct TransitionProperties :  
Sendable {  
  
    public init(hasMotion: Bool = true)  
        /// Whether the transition includes  
motion.  
        ///
```

```
    /// When this behavior is included in
a transition, that transition will be
    /// replaced by opacity when Reduce
Motion is enabled.
    ///
    /// Defaults to `true`.
public var hasMotion: Bool
}

/// A View created from a swift tuple of
View values.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct TupleView<T> : View
{

    public var value: T

    @inlinable public init(_ value: T)

        /// The type of view representing the
body of this view.
        ///
        /// When you create a custom view,
Swift infers this type from your
        /// implementation of the required
``View/body-swift.property`` property.
        @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
        public typealias Body = Never
}

/// Defines how typesetting language is
```

```
determined for text.  
///  
/// Use a modifier like  
``View/typesettingLanguage(_:isEnabled:)``  
  
/// to specify the typesetting language.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
public struct TypesettingLanguage :  
Sendable, Equatable {  
  
    /// Automatic language behavior.  
    ///  
    /// When determining the language to  
use for typesetting the current UI  
    /// language and preferred languages  
will be considered. For example, if  
    /// the current UI locale is for  
English and Thai is included in the  
    /// preferred languages then line  
heights will be taller to accommodate the  
    /// taller glyphs used by Thai.  
    public static let automatic:  
        TypesettingLanguage  
  
    /// Use explicit language.  
    ///  
    /// An explicit language will be used  
for typesetting. For example, if used  
    /// with Thai language the line  
heights will be as tall as needed to  
    /// accommodate Thai.  
    ///
```

```
    /// - Parameters:  
    ///   - language: The language to use  
for typesetting.  
    /// - Returns: A  
`TypesettingLanguage`.  
    public static func explicit(_  
language: Locale.Language) ->  
TypesettingLanguage  
  
    /// 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:  
TypesettingLanguage, b:  
TypesettingLanguage) -> Bool  
}  
  
/// A rectangular shape with rounded  
corners with different values, aligned  
/// inside the frame of the view  
containing it.  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
@frozen public struct
```

```
UnevenRoundedRectangle : Shape {  
  
    /// The radii of each corner of the  
    rounded rectangle.  
    public var cornerRadii:  
    RectangleCornerRadii  
  
    /// The style of corners drawn by the  
    rounded rectangle.  
    public var style: RoundedCornerStyle  
  
    /// Creates a new rounded rectangle  
    shape with uneven corners.  
    ///  
    /// - Parameters:  
    ///   - cornerRadii: the radii of  
    each corner.  
    ///   - style: the style of corners  
    drawn by the shape.  
    @inlinable public init(cornerRadii:  
    RectangleCornerRadii, style:  
    RoundedCornerStyle = .continuous)  
  
    /// Creates a new rounded rectangle  
    shape with uneven corners.  
    public init(topLeadingRadius: CGFloat  
= 0, bottomLeadingRadius: CGFloat = 0,  
bottomTrailingRadius: CGFloat = 0,  
topTrailingRadius: CGFloat = 0, style:  
RoundedCornerStyle = .continuous)  
  
    /// Describes this shape as a path  
    within a rectangular frame of reference.
```

```
///  
/// - Parameter rect: The frame of  
reference for describing this shape.  
///  
/// - Returns: A path that describes  
this shape.  
nonisolated public func path(in rect:  
CGRect) -> Path  
  
/// The data to animate.  
public var animatableData:  
RectangleCornerRadii.AnimatableData  
  
/// The type defining the data to  
animate.  
@available(iOS 16.0, tvOS 16.0,  
watchOS 9.0, macOS 13.0, *)  
public typealias AnimatableData =  
RectangleCornerRadii.AnimatableData  
  
/// The type of view representing the  
body of this view.  
///  
/// When you create a custom view,  
Swift infers this type from your  
/// implementation of the required  
``View/body-swift.property`` property.  
@available(iOS 16.0, tvOS 16.0,  
watchOS 9.0, macOS 13.0, *)  
public typealias Body  
}  
  
@available(iOS 16.0, macOS 13.0, tvOS
```

```
16.0, watchOS 9.0, *)
extension UnevenRoundedRectangle :  
InsettableShape {  
  
    /// Returns `self` inset by `amount`.  
    @inlinable nonisolated public func  
    inset(by amount: CGFloat) -> some  
    InsettableShape  
  
    /// The type of the inset shape.  
    @available(iOS 16.0, tvOS 16.0,  
    watchOS 9.0, macOS 13.0, *)  
    public typealias InsetShape = some  
    InsettableShape  
}  
  
/// A function defined by a two-  
dimensional curve that maps an input  
/// progress in the range [0,1] to an  
output progress that is also in the  
/// range [0,1]. By changing the shape of  
the curve, the effective speed  
/// of an animation or other  
interpolation can be changed.  
///  
/// The horizontal (x) axis defines the  
input progress: a single input  
/// progress value in the range [0,1]  
must be provided when evaluating a  
/// curve.  
///  
/// The vertical (y) axis maps to the
```

```
output progress: when a curve is
/// evaluated, the y component of the
point that intersects the input progress
/// is returned.
@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
public struct UnitCurve {

    /// Creates a new curve using bezier
control points.
    ///
    /// The x components of the control
points are clamped to the range [0,1]
when
    /// the curve is evaluated.
    ///
    /// - Parameters:
    ///   - startControlPoint: The cubic
Bézier control point associated with
        /// the curve's start point at
(0, 0). The tangent vector from the
        /// start point to its control
point defines the initial velocity of
        /// the timing function.
    ///   - endControlPoint: The cubic
Bézier control point associated with the
        /// curve's end point at (1, 1).
The tangent vector from the end point
        /// to its control point defines
the final velocity of the timing
        /// function.
    public static func
bezier(startControlPoint: UnitPoint,
```

```
endControlPoint: UnitPoint) -> UnitCurve
```

```
    /// Returns the output value (y  
component) of the curve at the given  
time.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - time: The input progress (x  
component). The provided value is
```

```
    ///     clamped to the range [0,1].
```

```
    ///
```

```
    /// - Returns: The output value (y  
component) of the curve at the given  
/// progress.
```

```
public func value(at progress:  
Double) -> Double
```

```
    /// Returns the rate of change (first  
derivative) of the output value of
```

```
    /// the curve at the given time.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - progress: The input progress  
(x component). The provided value is
```

```
    ///     clamped to the range [0,1].
```

```
    ///
```

```
    /// - Returns: The velocity of the  
output value (y component) of the curve  
/// at the given time.
```

```
public func velocity(at progress:  
Double) -> Double
```

```
    /// Returns a copy of the curve with
```

```
its x and y components swapped.  
///  
/// The inverse can be used to solve  
a curve in reverse: given a  
/// known output (y) value, the  
corresponding input (x) value can be  
found  
/// by using `inverse`:  
///  
///     let curve =  
UnitCurve.easeInOut  
///  
///     /// The input time for which  
an easeInOut curve returns 0.6.  
///     let inputTime =  
curve.inverse.evaluate(at: 0.6)  
///  
public var inverse: UnitCurve { get }  
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension UnitCurve : Sendable {  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension UnitCurve : Hashable {  
  
    /// 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: UnitCurve,
b: UnitCurve) -> 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 }
}

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

    /// A bezier curve that starts out
    slowly, speeds up over the middle, then
    /// slows down again as it approaches
    the end.
    ///
    /// The start and end control points
```

```
are located at (x: 0.42, y: 0) and
    /// (x: 0.58, y: 1).
    @available(*, deprecated, message:
"Use easeInOut instead")
    public static let easeInEaseOut:
UnitCurve

    /// A bezier curve that starts out
slowly, speeds up over the middle, then
    /// slows down again as it approaches
the end.
    ///
    /// The start and end control points
are located at (x: 0.42, y: 0) and
    /// (x: 0.58, y: 1).
    public static let easeInOut:
UnitCurve

    /// A bezier curve that starts out
slowly, then speeds up as it finishes.
    ///
    /// The start and end control points
are located at (x: 0.42, y: 0) and
    /// (x: 1, y: 1).
    public static let easeIn: UnitCurve

    /// A bezier curve that starts out
quickly, then slows down as it
    /// approaches the end.
    ///
    /// The start and end control points
are located at (x: 0, y: 0) and
    /// (x: 0.58, y: 1).
```

```
public static let easeOut: UnitCurve

    /// A curve that starts out slowly,
    then speeds up as it finishes.
    /**
     * The shape of the curve is equal
     * to the fourth (bottom right) quadrant
     * of a unit circle.
     */
    public static let circularEaseIn:
UnitCurve

    /// A circular curve that starts out
    quickly, then slows down as it
    /**
     * approaches the end.
     */
    /**
     * The shape of the curve is equal
     * to the second (top left) quadrant of
     * a unit circle.
     */
    public static let circularEaseOut:
UnitCurve

    /// A circular curve that starts out
    slowly, speeds up over the middle,
    /**
     * then slows down again as it
     * approaches the end.
     */
    /**
     * The shape of the curve is defined
     * by a piecewise combination of
     * `circularEaseIn` and
     * `circularEaseOut`.
     */
    public static let circularEaseInOut:
UnitCurve
```

```
    /// A linear curve.  
    ///  
    /// As the linear curve is a straight  
line from (0, 0) to (1, 1),  
    /// the output progress is always  
equal to the input progress, and  
    /// the velocity is always equal to  
1.0.  
    public static let linear: UnitCurve  
}
```

```
/// A normalized 2D point in a view's  
coordinate space.  
///  
/// Use a unit point to represent a  
location in a view without having to know  
/// the view's rendered size. The point  
stores a value in each dimension that  
/// indicates the fraction of the view's  
size in that dimension --- measured  
/// from the view's origin --- where the  
point appears. For example, you can  
/// create a unit point that represents  
the center of any view by using the  
/// value `0.5` for each dimension:  
///  
///     let unitPoint = UnitPoint(x: 0.5,  
y: 0.5)  
///  
/// To project the unit point into the  
rendered view's coordinate space,  
/// multiply each component of the unit  
point with the corresponding
```

```
/// component of the view's size:  
///  
///     let projectedPoint = CGPoint(  
///         x: unitPoint.x * size.width,  
///         y: unitPoint.y * size.height  
///     )  
///  
/// You can perform this calculation  
yourself if you happen to know a view's  
/// size, or if you want to use the unit  
point for some custom purpose, but  
/// SwiftUI typically does this for you  
to carry out operations that  
/// you request, like when you:  
///  
/// * Transform a shape using a shape  
modifier. For example, to rotate a  
/// shape with  
``Shape/rotation(_:anchor:)``, you  
indicate an anchor point  
/// that you want to rotate the shape  
around.  
/// * Override the alignment of the view  
in a ``Grid`` cell using the  
/// ``View/gridCellAnchor(_:)`` view  
modifier. The grid aligns the projection  
/// of a unit point onto the view with  
the projection of the same unit point  
/// onto the cell.  
/// * Create a gradient that has a  
center, or start and stop points,  
relative  
/// to the shape that you are styling.
```

See the gradient methods in  
/// ``ShapeStyle``.

///

/// You can create custom unit points  
with explicit values, like the example  
/// above, or you can use one of the  
built-in unit points that SwiftUI  
provides,  
/// like ``zero``, ``center``, or  
``topTrailing``. The built-in values  
/// correspond to the alignment positions  
of the similarly named, built-in  
/// ``Alignment`` types.

///

/// > Note: A unit point with one or more  
components outside the range `[0, 1]`  
/// projects to a point outside of the  
view.

///

/// **Layout direction**

///

/// When a person configures their device  
to use a left-to-right language like  
/// English, the system places the view's  
origin in its top-left corner,  
/// with positive x toward the right and  
positive y toward the bottom of the  
/// view. In a right-to-left environment,  
the origin moves to the upper-right  
/// corner, and the positive x direction  
changes to be toward the left. You  
/// don't typically need to do anything  
to handle this change, because SwiftUI

```
/// applies the change to all aspects of
the system. For example, see the
/// discussion about layout direction in
``HorizontalAlignment``.
///
/// It's important to test your app for
the different locales that you
/// distribute your app in. For more
information about the localization
process,
/// see
<doc://com.apple.documentation/documentation/Xcode/localization>.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct UnitPoint : Hashable {

    /// The normalized distance from the
origin to the point in the horizontal
    /// direction.
    public var x: CGFloat

    /// The normalized distance from the
origin to the point in the vertical
    /// dimension.
    public var y: CGFloat

    /// Creates a unit point at the
origin.
    ///
    /// A view's origin appears in the
top-left corner in a left-to-right
```

```
    /// language environment, with  
    positive x toward the right. It appears  
    in
```

```
    /// the top-right corner in a right-  
    to-left language, with positive x toward  
    /// the left. Positive y is always  
    toward the bottom of the view.
```

```
    @inlinable public init()
```

```
    /// Creates a unit point with the  
    specified horizontal and vertical  
    offsets.
```

```
    ///
```

```
    /// Values outside the range ` [0, 1]`  
    project to points outside of a view.
```

```
    ///
```

```
    /// - Parameters:
```

```
    /// - x: The normalized distance  
    from the origin to the point in the
```

```
    /// horizontal direction.
```

```
    /// - y: The normalized distance  
    from the origin to the point in the
```

```
    /// vertical direction.
```

```
    @inlinable public init(x: CGFloat, y:  
    CGFloat)
```

```
    /// The origin of a view.
```

```
    ///
```

```
    /// A view's origin appears in the  
    top-left corner in a left-to-right
```

```
    /// language environment, with  
    positive x toward the right. It appears  
    in
```

```
    /// the top-right corner in a right-to-left language, with positive x toward  
    /// the left. Positive y is always toward the bottom of the view.
```

```
    public static let zero: UnitPoint
```

```
    /// A point that's centered in a view.
```

```
    ///
```

```
    /// This point occupies the position where the horizontal and vertical
```

```
    /// alignment guides intersect for ``Alignment/center`` alignment.
```

```
    public static let center: UnitPoint
```

```
    /// A point that's centered vertically on the leading edge of a view.
```

```
    ///
```

```
    /// This point occupies the position where the horizontal and vertical
```

```
    /// alignment guides intersect for ``Alignment/leading`` alignment.
```

```
    /// The leading edge appears on the left in a left-to-right language
```

```
    /// environment and on the right in a right-to-left environment.
```

```
    public static let leading: UnitPoint
```

```
    /// A point that's centered vertically on the trailing edge of a view.
```

```
    ///
```

```
    /// This point occupies the position
```

```
where the horizontal and vertical
    /// alignment guides intersect for
``Alignment/trailing`` alignment.
    /// The trailing edge appears on the
right in a left-to-right language
    /// environment and on the left in a
right-to-left environment.
public static let trailing: UnitPoint

    /// A point that's centered
horizontally on the top edge of a view.
    ///
    /// This point occupies the position
where the horizontal and vertical
    /// alignment guides intersect for
``Alignment/top`` alignment.
public static let top: UnitPoint

    /// A point that's centered
horizontally on the bottom edge of a
view.
    ///
    /// This point occupies the position
where the horizontal and vertical
    /// alignment guides intersect for
``Alignment/bottom`` alignment.
public static let bottom: UnitPoint

    /// A point that's in the top,
leading corner of a view.
    ///
    /// This point occupies the position
where the horizontal and vertical
```

```
    /// alignment guides intersect for
``Alignment/topLeading`` alignment.
    /// The leading edge appears on the
left in a left-to-right language
    /// environment and on the right in a
right-to-left environment.
public static let topLeading:
UnitPoint
```

```
    /// A point that's in the top,
trailing corner of a view.
///
/// This point occupies the position
where the horizontal and vertical
    /// alignment guides intersect for
``Alignment/topTrailing`` alignment.
    /// The trailing edge appears on the
right in a left-to-right language
    /// environment and on the left in a
right-to-left environment.
public static let topTrailing:
UnitPoint
```

```
    /// A point that's in the bottom,
leading corner of a view.
///
/// This point occupies the position
where the horizontal and vertical
    /// alignment guides intersect for
``Alignment/bottomLeading`` alignment.
    /// The leading edge appears on the
left in a left-to-right language
    /// environment and on the right in a
```

right-to-left environment.

```
public static let bottomLeading:  
UnitPoint
```

```
    /// A point that's in the bottom,  
trailing corner of a view.
```

```
    ///
```

```
    /// This point occupies the position  
where the horizontal and vertical
```

```
    /// alignment guides intersect for  
``Alignment/bottomTrailing`` alignment.
```

```
    /// The trailing edge appears on the  
right in a left-to-right language
```

```
    /// environment and on the left in a  
right-to-left environment.
```

```
public static let bottomTrailing:  
UnitPoint
```

```
    /// 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: UnitPoint,  
b: UnitPoint) -> 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 }  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension UnitPoint : Animatable {  
  
    /// The type defining the data to  
animate.  
    public typealias AnimatableData =  
AnimatablePair<CGFloat, CGFloat>  
  
    /// The data to animate.  
    public var animatableData:  
UnitPoint.AnimatableData  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension UnitPoint : Sendable {  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
extension UnitPoint : BitwiseCopyable {  
}  
  
/// A set of values that indicate the  
visual size available to the view.  
///  
/// You receive a size class value when  
you read either the  
///  
``EnvironmentValues/horizontalSizeClass``  
or  
///  
``EnvironmentValues/verticalSizeClass``  
environment value. The value tells  
/// you about the amount of space  
available to your views in a given  
/// direction. You can read the size  
class like any other of the  
/// ``EnvironmentValues``, by creating a  
property with the ``Environment``  
/// property wrapper:  
///  
///  
@Environment(\.horizontalSizeClass)  
private var horizontalSizeClass  
///     @Environment(\.verticalSizeClass)  
private var verticalSizeClass  
///  
/// SwiftUI sets the size class based on  
several factors, including:  
///  
/// * The current device type.  
/// * The orientation of the device.
```

```
/// * The appearance of Slide Over and
Split View on iPad.
///
/// Several built-in views change their
behavior based on the size class.
/// For example, a ``NavigationView```
presents a multicolumn view when
/// the horizontal size class is
``UserInterfaceSizeClass/regular```,
/// but a single column otherwise. You
can also adjust the appearance of
/// custom views by reading the size
class and conditioning your views.
/// If you do, be prepared to handle size
class changes while
/// your app runs, because factors like
device orientation can change at
/// runtime.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public enum UserInterfaceSizeClass :  
Sendable {  
  
    /// The compact size class.
    case compact  
  
    /// The regular size class.
    case regular  
  
    /// 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:  
UserInterfaceSizeClass, b:  
UserInterfaceSizeClass) -> Bool

  /// 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)

    /// 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 }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension UserInterfaceSizeClass :
Equatable {

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension UserInterfaceSizeClass :
```

```
Hashable {  
}  
  
/// A view that arranges its subviews in  
a vertical line.  
///  
/// Unlike ``LazyVStack``, which only  
renders the views when your app needs to  
/// display them, a `VStack` renders the  
views all at once, regardless  
/// of whether they are on- or offscreen.  
Use the regular `VStack` when you have  
/// a small number of subviews or don't  
want the delayed rendering behavior  
/// of the "lazy" version.  
///  
/// The following example shows a simple  
vertical stack of 10 text views:  
///  
///     var body: some View {  
///         VStack(  
///             alignment: .leading,  
///             spacing: 10  
///         ) {  
///             ForEach(  
///                 1...10,  
///                 id: \.self  
///             ) {  
///                 Text("Item \($0)")  
///             }  
///         }  
///     }  
///
```

```
/// ! [Ten text views, named Item 1
through Item 10, arranged in a
/// vertical line.] (SwiftUI-VStack-
simple.png)
///
/// > Note: If you need a vertical stack
that conforms to the ``Layout``
/// protocol, like when you want to
create a conditional layout using
/// ``AnyLayout``, use ``VStackLayout``
instead.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct VStack<Content> :
View where Content : View {

    /// Creates an instance with the
given spacing and horizontal alignment.
    ///
    /// - Parameters:
    ///   - alignment: The guide for
aligning the subviews in this stack. This
    ///   guide has the same vertical
screen coordinate for every subview.
    ///   - spacing: The distance between
adjacent subviews, or `nil` if you
    ///   want the stack to choose a
default distance for each pair of
    ///   subviews.
    ///   - content: A view builder that
creates the content of this stack.
    @inlinable public init(alignment:
HorizontalAlignment = .center, spacing:
```

```
CGFloat? = nil, @ViewBuilder content: ()  
-> Content)  
  
    /// The type of view representing the  
body of this view.  
    ///  
    /// When you create a custom view,  
Swift infers this type from your  
    /// implementation of the required  
``View/body-swift.property`` property.  
    @available(iOS 13.0, tvOS 13.0,  
watchOS 6.0, macOS 10.15, *)  
    public typealias Body = Never  
}  
  
/// A vertical container that you can use  
in conditional layouts.  
///  
/// This layout container behaves like a  
``VStack``, but conforms to the  
/// ``Layout`` protocol so you can use it  
in the conditional layouts that you  
/// construct with ``AnyLayout``. If you  
don't need a conditional layout, use  
/// ``VStack`` instead.  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
@frozen public struct VStackLayout :  
Layout {  
  
    /// The horizontal alignment of  
subviews.  
    public var alignment:
```

## HorizontalAlignment

```
    /// The distance between adjacent
    subviews.
    ///
    /// Set this value to `nil` to use
    default distances between subviews.
    public var spacing: CGFloat?

    /// Creates a vertical stack with the
    specified spacing and horizontal
    /// alignment.
    ///
    /// - Parameters:
    ///   - alignment: The guide for
    aligning the subviews in this stack. It
    /// has the same horizontal
    screen coordinate for all subviews.
    /// - spacing: The distance
    between adjacent subviews. Set this value
    /// to `nil` to use default
    distances between subviews.
    @inlinable public init(alignment:
    HorizontalAlignment = .center, spacing:
    CGFloat? = nil)

    /// The type defining the data to
    animate.
    @available(iOS 16.0, tvOS 16.0,
    watchOS 9.0, macOS 13.0, *)
    public typealias AnimatableData =
    EmptyAnimatableData
```

```
    /// Cached values associated with the
    layout instance.
    ///
    /// If you create a cache for your
    custom layout, you can use
    /// a type alias to define this type
    as your data storage type.
    /// Alternatively, you can refer to
    the data storage type directly in all
    /// the places where you work with
    the cache.
    ///
    /// See ``makeCache(subviews:)`` for
    more information.
    @available(iOS 16.0, tvOS 16.0,
    watchOS 9.0, macOS 13.0, *)
    public typealias Cache
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension VStackLayout : Sendable {
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension VStackLayout : BitwiseCopyable
{
}

/// A type that can serve as the
animatable data of an animatable type.
///
```

```
/// `VectorArithmetic` extends the
`AdditiveArithmetic` protocol with scalar
/// multiplication and a way to query the
vector magnitude of the value. Use
/// this type as the `animatableData`
associated type of a type that conforms
to
/// the ``Animatable`` protocol.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
public protocol VectorArithmetic : AdditiveArithmetic {

    /// Multiplies each component of this
    value by the given value.
    mutating func scale(by rhs: Double)

    /// Returns the dot-product of this
    vector arithmetic instance with itself.
    var magnitudeSquared: Double { get }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension VectorArithmetic {

    /// Returns a value with each
    component of this value multiplied by the
    /// given value.
    public func scaled(by rhs: Double) ->
Self

    /// Interpolates this value with
```

```
`other` by the specified `amount`.  
    ///  
    /// This is equivalent to `self =  
    self + (other - self) * amount`.  
    public mutating func  
interpolate(towards other: Self, amount:  
Double)  
  
        /// Returns this value interpolated  
with `other` by the specified `amount`.  
        ///  
        /// This result is equivalent to  
`self + (other - self) * amount`.  
    public func interpolated(towards  
other: Self, amount: Double) -> Self  
}  
  
/// An alignment position along the  
vertical axis.  
///  
/// Use vertical alignment guides to  
position views  
/// relative to one another vertically,  
like when you place views side-by-side  
/// in an ``HStack`` or when you create a  
row of views in a ``Grid`` using  
/// ``GridRow``. The following example  
demonstrates common built-in  
/// vertical alignments:  
///  
/// ![Five rows of content. Each row  
contains text inside  
/// a box with horizontal lines to the
```

left and the right of the box. The  
/// lines are aligned vertically with the  
text in a different way for each  
/// row, corresponding to the content of  
the text in that row. The text strings  
/// are, in order, top, center, bottom,  
first text baseline, and last text  
/// baseline.] (VerticalAlignment-1-iOS)  
///  
/// You can generate the example above by  
creating a series of rows  
/// implemented as horizontal stacks,  
where you configure each stack with a  
/// different alignment guide:  
///  
/// private struct  
VerticalAlignmentGallery: View {  
/// var body: some View {  
/// VStack(spacing: 30) {  
/// row(alignment: .top,  
text: "Top")  
/// row(alignment: .center, text: "Center")  
/// row(alignment: .bottom, text: "Bottom")  
/// row(alignment: .firstTextBaseline, text:  
"First Text Baseline")  
/// row(alignment: .lastTextBaseline, text:  
"Last Text Baseline")  
/// }  
/// }  
/// }

```
///  
///     private func row(alignment:  
VerticalAlignment, text: String) -> some  
View {  
    ///             HStack(alignment:  
alignment, spacing: 0) {  
    ///  
Color.red.frame(height: 1)  
    ///  
Text(text).font(.title).border(.gray)  
    ///  
Color.red.frame(height: 1)  
    ///             }  
    ///             }  
    ///             }  
    ///  
    /// During layout, SwiftUI aligns the  
views inside each stack by bringing  
/// together the specified guides of the  
affected views. SwiftUI calculates  
/// the position of a guide for a  
particular view based on the  
characteristics  
/// of the view. For example, the  
``VerticalAlignment/center`` guide  
appears  
/// at half the height of the view. You  
can override the guide calculation for a  
/// particular view using the  
``View/alignmentGuide(_:computeValue:)``  
/// view modifier.  
///  
/// ### Text baseline alignment
```

```
///  
/// Use the  
``VerticalAlignment/firstTextBaseline``  
or  
///  
``VerticalAlignment/lastTextBaseline``  
guide to match the bottom of either  
/// the top- or bottom-most line of text  
that a view contains, respectively.  
/// Text baseline alignment excludes the  
parts of characters that descend  
/// below the baseline, like the tail on  
lower case g and j:  
///  
///  
row(alignment: .firstTextBaseline, text:  
"fghijkl")  
///  
/// If you use a text baseline alignment  
on a view that contains no text,  
/// SwiftUI applies the equivalent of  
``VerticalAlignment/bottom`` alignment  
/// instead. For the row in the example  
above, SwiftUI matches the bottom of  
/// the horizontal lines with the  
baseline of the text:  
///  
/// ! [A string containing the lowercase  
letters f, g, h, i, j, and  
/// k. The string is inside a box, and  
horizontal lines appear to the left and  
/// to the right of the box. The lines  
align with the bottom of the text,
```

```
/// excluding the descenders of letters g  
and j, which extend below the  
/// baseline.](VerticalAlignment-2-iOS)  
///  
/// Aligning a text view to its baseline  
rather than to the bottom of its frame  
/// produces the best layout effect in  
many cases, like when creating forms.  
/// For example, you can align the  
baseline of descriptive text in  
/// one ``GridRow`` cell with the  
baseline of a text field, or the label  
/// of a checkbox, in another cell in the  
same row.  
///  
/// Custom alignment guides  
///  
/// You can create a custom vertical  
alignment guide by first creating a type  
/// that conforms to the ``AlignmentID``  
protocol, and then using that type to  
/// initialize a new static property on  
`VerticalAlignment`:  
///  
///     private struct  
FirstThirdAlignment: AlignmentID {  
///         static func defaultValue(in  
context: ViewDimensions) -> CGFloat {  
///             context.height / 3  
///         }  
///     }  
///  
///     extension VerticalAlignment {
```

```
///         static let firstThird =
VerticalAlignment(FirstThirdAlignment.self)
///
///
/// You implement the
``AlignmentID/defaultValue(in:)`` method
to calculate
/// a default value for the custom
alignment guide. The method receives a
/// ``ViewDimensions`` instance that you
can use to calculate a
/// value based on characteristics of the
view. The example above places
/// the guide at one-third of the height
of the view as measured from the
/// view's origin.
///
///
/// You can then use the custom alignment
guide like any built-in guide. For
/// example, you can use it as the
`alignment` parameter to an ``HStack``,
/// or to alter the guide calculation for
a specific view using the
///
``View/alignmentGuide(_:computeValue:)``
view modifier.
///
/// ### Composite alignment
///
/// Combine a `VerticalAlignment` with a
``HorizontalAlignment`` to create a
/// composite ``Alignment`` that
```

indicates both vertical and horizontal  
/// positioning in one value. For  
example, you could combine your custom  
/// `firstThird` vertical alignment from  
the previous section with a built-in  
/// ``HorizontalAlignment/center``  
horizontal alignment to use in a  
``ZStack``:

```
///  
///     struct LayeredHorizontalStripes:  
View {  
    ///         var body: some View {  
    ///             ZStack(alignment:  
Alignment(horizontal: .center,  
vertical: .firstThird)) {  
    ///                 horizontalStripes(color: .blue)  
    ///                     .frame(width:  
180, height: 90)  
    ///                 horizontalStripes(color: .green)  
    ///                     .frame(width: 70,  
height: 60)  
    ///             }  
    ///         }  
    ///     }  
  
    ///         private func  
horizontalStripes(color: Color) -> some  
View {  
    ///             VStack(spacing: 1) {  
    ///                 ForEach(0..<3) { _ in  
color }  
    ///             }  
}
```

```
    }
}
/// The example above uses widths and
/// heights that generate two mismatched
/// sets of three vertical stripes. The
/// ``ZStack`` centers the two sets
/// horizontally and aligns them
/// vertically one-third from the top
/// of each set. This aligns the bottom
/// edges of the top stripe from each set:
///
/// ! [Two sets of three vertically
/// stacked rectangles. The first
/// set is blue. The second set of
/// rectangles are green, smaller, and
/// layered
/// on top of the first set. The two sets
/// are centered horizontally, but align
/// vertically at the bottom edge of each
/// set's top-most
/// rectangle.] (VerticalAlignment-3-iOS)
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct VerticalAlignment : Equatable {

    /// Creates a custom vertical
    /// alignment of the specified type.
    ///
    /// Use this initializer to create a
    /// custom vertical alignment. Define
    /// an ``AlignmentID`` type, and then
```

```
use that type to create a new
    /// static property on
``VerticalAlignment``:
    /**
     /// private struct
FirstThirdAlignment: AlignmentID {
    /**
     static func
defaultValue(in context: ViewDimensions)
-> CGFloat {
    /**
         context.height / 3
    /**
        }
    /**
        }
    /**
    /**
        extension VerticalAlignment {
    /**
        static let firstThird =
VerticalAlignment(FirstThirdAlignment.self)
    /**
        }
    /**
    /**
        /// Every vertical alignment instance
that you create needs a unique
        /// identifier. For more information,
see ``AlignmentID``.
    /**
    /**
        /// - Parameter id: The type of an
identifier that uniquely identifies a
        /// vertical alignment.
    public init(_ id: any
AlignmentID.Type)

    /**
        You don't use this property
directly.
    public let key: AlignmentKey
```

```
    /// 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:  
VerticalAlignment, b: VerticalAlignment)  
-> Bool  
}
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension VerticalAlignment {  
  
    /// Merges a sequence of explicit  
    alignment values produced by  
    /// this instance.  
    ///  
    /// For most alignment types, this  
    method returns the mean of all non-`nil`  
    /// values. However, some types use  
    other rules. For example,  
    ///  
    ``VerticalAlignment/firstTextBaseline``  
    returns the minimum value,
```

```
    /// while
``VerticalAlignment/lastTextBaseline``
returns the maximum value.
    public func combineExplicit<S>(_
values: S) -> CGFloat? where S :
Sequence, S.Element == CGFloat?
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension VerticalAlignment {

    /// A guide that marks the top edge
of the view.
    ///
    /// Use this guide to align the top
edges of views:
    ///
    /// ![A box that contains the word,
Top. A horizontal
    /// line appears on either side of
the box. The lines align vertically
    /// with the top edge of the box.]
```

(VerticalAlignment-top-1-iOS)

```
    ///
    /// The following code generates the
image above using an ``HStack``:
```

```
    ///
    /// struct VerticalAlignmentTop:
View {
    ///
        var body: some View {
    ///
HStack(alignment: .top, spacing: 0) {
```

```
    /**
Color.red.frame(height: 1)
    /**
Text("Top").font(.title).border(.gray)
    /**
Color.red.frame(height: 1)
    /**
}
    /**
}
    /**
}
    /**
public static let top:
VerticalAlignment
```

```
    /**
A guide that marks the vertical
center of the view.
    /**
    /**
Use this guide to align the
centers of views:
    /**
    /**
! [A box that contains the word,
Center. A horizontal
    /**
line appears on either side of
the box. The lines align vertically
    /**
with the center of the box.]
```

(VerticalAlignment-center-1-iOS)

```
    /**
    /**
The following code generates the
image above using an ``HStack``:
```

```
    /**
    /**
struct
VerticalAlignmentCenter: View {
    /**
        var body: some View {
    /**

```

```
HStack(alignment: .center, spacing: 0) {
    /**
     Color.red.frame(height: 1)
    /**
     Text("Center").font(.title).border(.gray)
    /**
     Color.red.frame(height: 1)
    /**
     */
    /**
     */
    /**
     */
    /**
     public static let center:
VerticalAlignment

    /**
     A guide that marks the bottom
edge of the view.
    /**
    /**
     Use this guide to align the
bottom edges of views:
    /**
    /**
     ! [A box that contains the word,
Bottom. A horizontal
    /**
     line appears on either side of
the box. The lines align vertically
    /**
     with the bottom edge of the box.] 
(VerticalAlignment-bottom-1-iOS)
    /**
    /**
     The following code generates the
image above using an ``HStack``:
    /**
    /**
     struct
VerticalAlignmentBottom: View {
    /**
         var body: some View {
```

```
    /**
HStack(alignment: .bottom, spacing: 0) {
    /**
Color.red.frame(height: 1)
    /**
Text("Bottom").font(.title).border(.gray)
    /**
Color.red.frame(height: 1)
    /**
}
    /**
}
    /**
}
    /**
public static let bottom:
VerticalAlignment
```

```
    /**
     * A guide that marks the top-most
text baseline in a view.
    /**
     * Use this guide to align with the
baseline of the top-most text in a
    /**
view. The guide aligns with the
bottom of a view that contains no text:
    /**
     * ! [A box that contains the text,
First Text Baseline.
    /**
     * A horizontal line appears on
either side of the box. The lines align
    /**
vertically with the baseline of
the first line of
    /**
text.](VerticalAlignment-
firstTextBaseline-1-iOS)
    /**
     * The following code generates the
```

image above using an ``HStack``:

```
///
/// struct
VerticalAlignmentFirstTextBaseline: View
{
    ///
    var body: some View {
        ///
        HStack(alignment: .firstTextBaseline,
               spacing: 0) {
            ///
            Color.red.frame(height: 1)
                ///
                Text("First Text
Baseline").font(.title).border(.gray)
            ///
            Color.red.frame(height: 1)
                ///
                }
            ///
            }
        ///
        public static let firstTextBaseline:
VerticalAlignment
```

    /// A guide that marks the bottom-most text baseline in a view.

```
    ///
    /// Use this guide to align with the
    baseline of the bottom-most text in a
    /// view. The guide aligns with the
    bottom of a view that contains no text.
```

```
    ///
    /// ! [A box that contains the text,
    Last Text Baseline.
```

```
    /// A horizontal line appears on
    either side of the box. The lines align
```

```
    /// vertically with the baseline of
the last line of
    /// text.](VerticalAlignment-
lastTextBaseline-1-iOS)
    ///
    /// The following code generates the
image above using an ``HStack``:
    ///
    /// struct
VerticalAlignmentLastTextBaseline: View {
    ///         var body: some View {
    ///
HStack(alignment: .lastTextBaseline,
spacing: 0) {
    ///
Color.red.frame(height: 1)
    ///             Text("Last Text
Baseline").font(.title).border(.gray)
    ///
Color.red.frame(height: 1)
    ///             }
    ///
    ///             }
    ///
    ///             }
    ///
public static let lastTextBaseline:
VerticalAlignment
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension VerticalAlignment : Sendable {
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension VerticalAlignment :  
BitwiseCopyable {  
}  
  
/// A direction on the horizontal axis.  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
@frozen public enum VerticalDirection :  
Int8, CaseIterable, Codable {  
  
    /// The upward direction.  
    case up  
  
    /// The downward direction.  
    case down  
  
    /// An efficient set of vertical  
    directions.  
    @frozen public struct Set : OptionSet {  
  
        /// The element type of the  
        option set.  
        ///  
        /// To inherit all the default  
        implementations from the `OptionSet`  
        protocol,  
        /// the `Element` type must be  
        `Self`, the default.  
        public typealias Element =  
VerticalDirection.Set
```

```
    /// The corresponding value of
the raw type.
    ///
    /// A new instance initialized
with `rawValue` will be equivalent to
this
    /// instance. For example:
    ///
    ///     enum PaperSize: String {
    ///         case A4, A5, Letter,
Legal
    ///     }
    ///
    ///     let selectedSize =
PaperSize.Letter
    ///
print(selectedSize.rawValue)
    /// // Prints "Letter"
    ///
    /// print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    /// // Prints "true"
public let rawValue: Int8

    /// Creates a new option set from
the given raw value.
    ///
    /// This initializer always
succeeds, even if the value passed as
`rawValue`
    /// exceeds the static properties
```

declared as part of the option set. This  
    /// example creates an instance  
of `ShippingOptions` with a raw value  
beyond

    /// the highest element, with a  
bit mask that effectively contains all  
the

```
    /// declared static members.  
    ///  
    ///     let extraOptions =  
ShippingOptions(rawValue: 255)  
    ///  
print(extraOptions.isStrictSuperset(of: .  
all))  
    ///     // Prints "true"  
    ///  
    /// - Parameter rawValue: The raw  
value of the option set to create. Each  
bit  
    ///     of `rawValue` potentially  
represents an element of the option set,  
    ///     though raw values may  
include bits that are not defined as  
distinct  
    ///     values of the `OptionSet`  
type.  
public init(rawValue: Int8)  
  
    /// A set containing only the  
leading horizontal direction.  
public static let up:  
VerticalDirection.Set
```

```
        /// A set containing only the
        trailing horizontal direction.
        public static let down:
VerticalDirection.Set

        /// A set containing the upward
and downward vertical directions.
        public static let all:
VerticalDirection.Set

        /// Creates a set of directions
containing only the specified direction.
        public init(_ direction:
VerticalDirection)

        /// The type of the elements of
an array literal.
        @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
        public typealias
ArrayLiteralElement =
VerticalDirection.Set.Element

        /// The raw type that can be used
to represent all values of the conforming
        /// type.
        ///
        /// Every distinct value of the
conforming type has a corresponding
unique
        /// value of the `RawValue` type,
but there may be values of the `RawValue`
```

```
        /// type that don't have a
        // corresponding value of the conforming
        // type.
        @available(iOS 18.0, tvOS 18.0,
watchOS 11.0, visionOS 2.0, macOS 15.0,
*)
    public typealias RawValue = Int8
}

/// Creates a new instance with the
// specified raw value.
///
/// If there is no value of the type
// that corresponds with the specified raw
/// value, this initializer returns
`nil`. For example:
///
///     enum PaperSize: String {
///         case A4, A5, Letter,
Legal
///     }
///
///     print(PaperSize(rawValue:
"Legal"))
///     // Prints
"Optional("PaperSize.Legal")"
///
///     print(PaperSize(rawValue:
"Tabloid"))
///     // Prints "nil"
///
/// - Parameter rawValue: The raw
// value to use for the new instance.
```

```
public init?(rawValue: Int8)

    /// A type that can represent a
    collection of all values of this type.
    @available(iOS 18.0, tvOS 18.0,
    watchOS 11.0, visionOS 2.0, macOS 15.0,
    *)
    public typealias AllCases =
    [VerticalDirection]

    /// The raw type that can be used to
    represent all values of the conforming
    /// type.
    ///
    /// Every distinct value of the
    conforming type has a corresponding
    unique
    /// value of the `RawValue` type, but
    there may be values of the `RawValue`
    /// type that don't have a
    corresponding value of the conforming
    type.
    @available(iOS 18.0, tvOS 18.0,
    watchOS 11.0, visionOS 2.0, macOS 15.0,
    *)
    public typealias RawValue = Int8

    /// A collection of all values of
    this type.
    nonisolated public static var
    allCases: [VerticalDirection] { get }

    /// The corresponding value of the
```

```
raw type.  
    ///  
    /// A new instance initialized with  
`.rawValue` will be equivalent to this  
    /// instance. For example:  
    ///  
    ///     enum PaperSize: String {  
    ///         case A4, A5, Letter,  
Legal  
    ///     }  
    ///  
    ///     let selectedSize =  
PaperSize.Letter  
    ///     print(selectedSize.rawValue)  
    ///     // Prints "Letter"  
    ///  
    ///     print(selectedSize ==  
PaperSize(rawValue:  
selectedSize.rawValue)!)  
    ///     // Prints "true"  
    public var rawValue: Int8 { get }  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension VerticalDirection : Equatable {  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension VerticalDirection : Hashable {  
}
```

```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension VerticalDirection :  
RawRepresentable {  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension VerticalDirection : Sendable {  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension VerticalDirection :  
BitwiseCopyable {  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension VerticalDirection.Set :  
Sendable {  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension VerticalDirection.Set :  
BitwiseCopyable {  
}  
  
/// An edge on the vertical axis.  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
@frozen public enum VerticalEdge : Int8,
```

```
CaseIterable, Codable {

    /// The top edge.
    case top

    /// The bottom edge.
    case bottom

    /// An efficient set of vertical
    edges.
    @frozen public struct Set : OptionSet
{

        /// The element type of the
        option set.
        ///
        /// To inherit all the default
        implementations from the `OptionSet`
        protocol,
        /// the `Element` type must be
        `Self`, the default.
        public typealias Element =
VerticalEdge.Set

        /// The corresponding value of
        the raw type.
        ///
        /// A new instance initialized
        with `rawValue` will be equivalent to
        this
        /// instance. For example:
        ///
        ///     enum PaperSize: String {
```

```
        /// case A4, A5, Letter,
Legal
        ///
        ///
        /// let selectedSize =
PaperSize.Letter
        ///
print(selectedSize.rawValue)
        /// // Prints "Letter"
        ///
        /// print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
        /// // Prints "true"
public let rawValue: Int8

        /// Creates a new option set from
the given raw value.
        ///
        /// This initializer always
succeeds, even if the value passed as
`rawValue`
        /// exceeds the static properties
declared as part of the option set. This
        /// example creates an instance
of `ShippingOptions` with a raw value
beyond
        /// the highest element, with a
bit mask that effectively contains all
the
        /// declared static members.
        ///
        /// let extraOptions =
```

```
ShippingOptions(rawValue: 255)
    /**
print(extraOptions.isStrictSuperset(of: .
all))
    /**
        // Prints "true"
    /**
        /**
         /// - Parameter rawValue: The raw
value of the option set to create. Each
bit
        /**
            of `rawValue` potentially
represents an element of the option set,
        /**
            though raw values may
include bits that are not defined as
distinct
        /**
            values of the `OptionSet`
type.
public init(rawValue: Int8)

        /**
            A set containing only the top
vertical edge.
public static let top:
VerticalEdge.Set

        /**
            A set containing only the
bottom vertical edge.
public static let bottom:
VerticalEdge.Set

        /**
            A set containing the top and
bottom vertical edges.
public static let all:
VerticalEdge.Set
```

```
    /// Creates a set of edges
    containing only the specified vertical
    edge.
    public init(_ e: VerticalEdge)

        /// The type of the elements of
        an array literal.
        @available(iOS 15.0, tvOS 15.0,
        watchOS 8.0, macOS 12.0, *)
        public typealias
        ArrayLiteralElement =
        VerticalEdge.Set.Element

            /// The raw type that can be used
            to represent all values of the conforming
            /// type.
            ///
            /// Every distinct value of the
            conforming type has a corresponding
            unique
            /// value of the `RawValue` type,
            but there may be values of the `RawValue`
            /// type that don't have a
            corresponding value of the conforming
            type.
            @available(iOS 15.0, tvOS 15.0,
            watchOS 8.0, macOS 12.0, *)
            public typealias RawValue = Int8
        }

        /// Creates a new instance with the
        specified raw value.
        ///
```

```
    /// If there is no value of the type
    /// that corresponds with the specified raw
    /// value, this initializer returns
    `nil`. For example:
    /**
     * enum PaperSize: String {
     *   case A4, A5, Letter,
     Legal
     */
    /**
     * print(PaperSize(rawValue:
     "Legal"))
     // Prints
     "Optional("PaperSize.Legal")"
     /**
     * print(PaperSize(rawValue:
     "Tabloid"))
     // Prints "nil"
     /**
     * - Parameter rawValue: The raw
     value to use for the new instance.
    public init?(rawValue: Int8)

    /// A type that can represent a
    collection of all values of this type.
    @available(iOS 15.0, tvOS 15.0,
    watchOS 8.0, macOS 12.0, *)
    public typealias AllCases =
    [VerticalEdge]

    /// The raw type that can be used to
    represent all values of the conforming
    /// type.
```

```
///  
/// Every distinct value of the  
conforming type has a corresponding  
unique  
    /// value of the `RawValue` type, but  
there may be values of the `RawValue`  
    /// type that don't have a  
corresponding value of the conforming  
type.  
@available(iOS 15.0, tvOS 15.0,  
watchOS 8.0, macOS 12.0, *)  
public typealias RawValue = Int8  
  
    /// A collection of all values of  
this type.  
nonisolated public static var  
allCases: [VerticalEdge] { get }  
  
    /// The corresponding value of the  
raw type.  
    ///  
    /// A new instance initialized with  
`rawValue` will be equivalent to this  
    /// instance. For example:  
    ///  
    ///     enum PaperSize: String {  
    ///         case A4, A5, Letter,  
Legal  
    ///     }  
    ///  
    ///     let selectedSize =  
PaperSize.Letter  
    ///     print(selectedSize.rawValue)
```

```
    ///      // Prints "Letter"
    ///
    ///      print(selectedSize ==
PaperSize(rawValue:
selectedSize.rawValue)!)
    ///      // Prints "true"
public var rawValue: Int8 { get }
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension VerticalEdge : Equatable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension VerticalEdge : Hashable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension VerticalEdge : RawRepresentable
{
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension VerticalEdge : Sendable {
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension VerticalEdge : BitwiseCopyable
```

```
{  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension VerticalEdge.Set : Sendable {  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension VerticalEdge.Set :  
BitwiseCopyable {  
}  
  
/// A type that represents part of your  
/// app's user interface and provides  
/// modifiers that you use to configure  
views.  
///  
/// You create custom views by declaring  
types that conform to the `View`  
/// protocol. Implement the required  
``View/body-swift.property`` computed  
/// property to provide the content for  
your custom view.  
///  
///     struct MyView: View {  
///         var body: some View {  
///             Text("Hello, World!")  
///         }  
///     }  
///  
/// Assemble the view's body by combining
```

one or more of the built-in views  
/// provided by SwiftUI, like the  
``Text`` instance in the example above,  
plus  
/// other custom views that you define,  
into a hierarchy of views. For more  
/// information about creating custom  
views, see <doc:Declaring-a-Custom-View>.  
///  
/// The `View` protocol provides a set of  
modifiers – protocol  
/// methods with default implementations  
– that you use to configure  
/// views in the layout of your app.  
Modifiers work by wrapping the  
/// view instance on which you call them  
in another view with the specified  
/// characteristics, as described in  
<doc:Configuring-Views>.  
/// For example, adding the  
``View-opacity(\_:)`` modifier to a  
/// text view returns a new view with  
some amount of transparency:  
///  
///     Text("Hello, World!")  
///         .opacity(0.5) // Display  
partially transparent text.  
///  
/// The complete list of default  
modifiers provides a large set of  
controls  
/// for managing views.  
/// For example, you can fine tune

```
<doc:View-Layout>,
/// add <doc:View-Accessibility>
information,
/// and respond to <doc:View-Input-and-
Events>.
/// You can also collect groups of
default modifiers into new,
/// custom view modifiers for easy reuse.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@MainActor @preconcurrency public
protocol View {
    /// The type of view representing the
body of this view.
    ///
    /// When you create a custom view,
Swift infers this type from your
    /// implementation of the required
``View/body-swift.property`` property.
    associatedtype Body : View

    /// The content and behavior of the
view.
    ///
    /// When you implement a custom view,
you must implement a computed
    /// `body` property to provide the
content for your view. Return a view
    /// that's composed of built-in views
that SwiftUI provides, plus other
    /// composite views that you've
already defined:
```

```
///  
///     struct MyView: View {  
///         var body: some View {  
///             Text("Hello, World!")  
///         }  
///     }  
///  
///     /// For more information about  
composing views and a view hierarchy,  
/// see <doc:Declaring-a-Custom-  
View>.  
@ViewBuilder @MainActor  
@preconcurrency var body: Self.Body { get  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Adds a condition that controls  
    whether users can interact with this  
    /// view.  
    ///  
    /// The higher views in a view  
    hierarchy can override the value you set  
    on  
    /// this view. In the following  
    example, the button isn't interactive  
    /// because the outer `disabled(_:)`  
    modifier overrides the inner one:  
    ///  
    ///     HStack {
```

```
    /**
     * - Parameter disabled: A Boolean value that determines whether users can interact with this view.
     */
    /**
     * - Returns: A view that controls whether users can interact with this view.
     */
    @inlinable nonisolated public func disabled(_ disabled: Bool) -> some View
```

}

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension View {
```

/// Offset this view by the horizontal and vertical amount specified in the

/// offset parameter.

///

/// Use `offset(\_:)` to shift the displayed contents by the amount

/// specified in the `offset` parameter.

///

/// The original dimensions of the view aren't changed by offsetting the

```
    /// contents; in the example below
    the gray border drawn by this view
    /// surrounds the original position
    of the text:
    /**
     /**
      Text("Offset by passing
      CGSize())
      /**
       .border(Color.green)
      /**
       .offset(CGSize(width: 20,
      height: 25))
      /**
       .border(Color.gray)
      /**
      /**
      /**
      ! [A screenshot showing a view
      that offset from its original position a
      /**
      CGPoint to specify the x and y
      offset.] (SwiftUI-View-offset.png)
      /**
      /**
      - Parameter offset: The distance
      to offset this view.
      /**
      /**
      - Returns: A view that offsets
      this view by `offset`.
      @inlinable nonisolated public func
      offset(_ offset: CGSize) -> some View
```

```
    /**
      Offset this view by the specified
      horizontal and vertical distances.
      /**
      /**
      Use `offset(x:y:)` to shift the
      displayed contents by the amount
      /**
      specified in the `x` and `y`
      parameters.
```

```
///  
/// The original dimensions of the  
view aren't changed by offsetting the  
/// contents; in the example below  
the gray border drawn by this view  
/// surrounds the original position  
of the text:  
///  
///     Text("Offset by passing  
horizontal & vertical distance")  
///             .border(Color.green)  
///             .offset(x: 20, y: 50)  
///             .border(Color.gray)  
///  
///     /// ! [A screenshot showing a view  
that offset from its original position  
/// using and x and y offset.]  
(swiftui-offset-xy.png)  
///  
/// - Parameters:  
///   - x: The horizontal distance to  
offset this view.  
///   - y: The vertical distance to  
offset this view.  
///  
/// - Returns: A view that offsets  
this view by `x` and `y`.  
@inlinable nonisolated public func  
offset(x: CGFloat = 0, y: CGFloat = 0) ->  
some View  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Fixes this view at its ideal size  
    // in the specified dimensions.  
    ///  
    /// This function behaves like  
    ``View/fixedSize()``, except with  
    /// `fixedSize(horizontal:vertical:)`  
    the fixing of the axes can be  
    /// optionally specified in one or  
    both dimensions. For example, if you  
    /// horizontally fix a text view  
    before wrapping it in the frame view,  
    /// you're telling the text view to  
    maintain its ideal _width_. The view  
    /// calculates this to be the space  
    needed to represent the entire string.  
    ///  
    ///     Text("A single line of text,  
    too long to fit in a box.")  
    ///             .fixedSize(horizontal:  
    true, vertical: false)  
    ///             .frame(width: 200,  
    height: 200)  
    ///             .border(Color.gray)  
    ///  
    /// This can result in the view  
    exceeding the parent's bounds, which may  
    or  
    /// may not be the effect you want.  
    ///
```

```
    /// ! [A screenshot showing a text
view exceeding the bounds of its
    /// parent.] (SwiftUI-View-
fixedSize-3.png)
    ///
    /// - Parameters:
    ///   - horizontal: A Boolean value
that indicates whether to fix the width
    ///   of the view.
    ///   - vertical: A Boolean value
that indicates whether to fix the height
    ///   of the view.
    ///
    /// - Returns: A view that fixes this
view at its ideal size in the
    /// dimensions specified by
`horizontal` and `vertical`.
@inlinable nonisolated public func
fixedSize(horizontal: Bool, vertical:
Bool) -> some View
```

```
    /// Fixes this view at its ideal
size.
    ///
    /// During the layout of the view
hierarchy, each view proposes a size to
    /// each child view it contains. If
the child view doesn't need a fixed size
    /// it can accept and conform to the
size offered by the parent.
    ///
    /// For example, a ``Text`` view
```

```
placed in an explicitly sized frame wraps
    /// and truncates its string to
remain within its parent's bounds:
    /**
     /**
      Text("A single line of text,
too long to fit in a box.")
      /**
           .frame(width: 200,
height: 200)
      /**
           .border(Color.gray)
      /**
      /**
        /**
         ! [A screenshot showing the text
in a text view contained within its
         /**
          parent.] (SwiftUI-View-
fixedSize-1.png)
      /**
      /**
        /**
         The `fixedSize()` modifier can be
used to create a view that maintains
        /**
         the *ideal size* of its children
both dimensions:
    /**
    /**
      Text("A single line of text,
too long to fit in a box.")
      /**
           .fixedSize()
      /**
           .frame(width: 200,
height: 200)
      /**
           .border(Color.gray)
      /**
      /**
        /**
         This can result in the view
exceeding the parent's bounds, which may
or
        /**
         may not be the effect you want.
        /**
        /**
          ! [A screenshot showing a text
```

```
view exceeding the bounds of its
    /// parent.](SwiftUI-View-
fixedSize-2.png)
    /**
     /// You can think of `fixedSize()` as
the creation of a *counter proposal*
     /// to the view size proposed to a
view by its parent. The ideal size of a
     /// view, and the specific effects of
`fixedSize()` depends on the
     /// particular view and how you have
configured it.
    /**
     /// To create a view that fixes the
view's size in either the horizontal or
     /// vertical dimensions, see
``View/fixedSize(horizontal:vertical:)``.
    /**
     /// - Returns: A view that fixes this
view at its ideal size.
    @inlinable nonisolated public func
fixedSize() -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Rotates a view's rendered output
in two dimensions around the specified
     /// point.
    /**

```

```
    /// This modifier rotates the view's
    /// content around the axis that points
    /// out of the xy-plane. It has no
    /// effect on the view's frame.
    /// The following code rotates text
    /// by 22° and then draws a border around
    /// the modified view to show that
    /// the frame remains unchanged by the
    /// rotation modifier:
    ///
    ///     Text("Rotation by passing an
    /// angle in degrees")
    ///         .rotationEffect(.degrees(
22))
    ///         .border(Color.gray)
    ///
    /// !-[A screenshot of text and a wide
    /// grey box. The text says Rotation by
    /// passing an angle in degrees. The baseline
    /// of the text is rotated clockwise by 22
    /// degrees relative to the box. The center
    /// of the box and the center of the text are
    /// aligned.] (SwiftUI-View-rotationEffect)
    ///
    /// - Parameters:
    ///   - angle: The angle by which to
    ///     rotate the view.
    ///   - anchor: A unit point within
    ///     the view about which to
    ///     perform the rotation. The
    ///     default value is ``UnitPoint/center``.
    /// - Returns: A view with rotated
    ///     content.
```

```
    @inlinable nonisolated public func
rotationEffect(_ angle: Angle, anchor:
UnitPoint = .center) -> some View
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Sets the color of the foreground
elements displayed by this view.
    ///
    /// - Parameter color: The foreground
color to use when displaying this
    /// view. Pass `nil` to remove any
custom foreground color and to allow
    /// the system or the container to
provide its own foreground color.
    /// If a container-specific
override doesn't exist, the system uses
    /// the primary color.
    ///
    /// - Returns: A view that uses the
foreground color you supply.
    @available(iOS, introduced: 13.0,
deprecated: 100000.0, renamed:
"foregroundStyle(_:)")
    @available(macOS, introduced: 10.15,
deprecated: 100000.0, renamed:
"foregroundStyle(_:)")
    @available(tvOS, introduced: 13.0,
deprecated: 100000.0, renamed:
```

```
"foregroundStyle(_:)")  
    @available(watchOS, introduced: 6.0,  
deprecated: 100000.0, renamed:  
"foregroundStyle(_:)")  
    @available(visionOS, introduced: 1.0,  
deprecated: 100000.0, renamed:  
"foregroundStyle(_:)")  
    @inlinable nonisolated public func  
foregroundColor(_ color: Color?) -> some  
View  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Sets the accent color for this  
view and the views it contains.  
    ///  
    /// Use `accentColor(_:)` when you  
want to apply a broad theme color to  
    /// your app's user interface. Some  
styles of controls use the accent color  
    /// as a default tint color.  
    ///  
    /// > Note: In macOS, SwiftUI applies  
customization of the accent color  
    /// only if the user chooses  
Multicolor under General > Accent color  
    /// in System Preferences.  
    ///  
    /// In the example below, the outer
```

```
``VStack`` contains two child views. The
    /// first is a button with the
default accent color. The second is a
``VStack``
    /// that contains a button and a
slider, both of which adopt the purple
    /// accent color of their containing
view. Note that the ``Text`` element
    /// used as a label alongside the
`Slider` retains its default color.
    ///
    ///     VStack(spacing: 20) {
    ///         Button(action: {}) {
    ///             Text("Regular
Button")
    ///         }
    ///         VStack {
    ///             Button(action: {}) {
    ///                 Text("Accented
Button")
    ///             }
    ///             HStack {
    ///                 Text("Accented
Slider")
    ///                 Slider(value:
$sliderValue, in: -100...100, step: 0.1)
    ///             }
    ///         }
    ///         .accentColor(.purple)
    ///     }
    ///
    /// ! [A VStack showing two child
views: one VStack containing a default
```

```
    /// accented button, and a second
VStack where the VStack has a purple
    /// accent color applied. The accent
color modifies the enclosed button and
    /// slider, but not the color of a
Text item used as a label for the
    /// slider.] (View-accentColor-1)
    ///
    /// - Parameter accentColor: The
color to use as an accent color. Set the
    /// value to `nil` to use the
inherited accent color.
    @available(iOS, introduced: 13.0,
deprecated: 100000.0, message: "Use the
asset catalog's accent color or
View.tint(_:) instead.")
    @available(macOS, introduced: 10.15,
deprecated: 100000.0, message: "Use the
asset catalog's accent color or
View.tint(_:) instead.")
    @available(tvOS, introduced: 13.0,
deprecated: 100000.0, message: "Use the
asset catalog's accent color or
View.tint(_:) instead.")
    @available(watchOS, introduced: 6.0,
deprecated: 100000.0, message: "Use the
asset catalog's accent color or
View.tint(_:) instead.")
    @available(visionOS, introduced: 1.0,
deprecated: 100000.0, message: "Use the
asset catalog's accent color or
View.tint(_:) instead.")
    @inlinable nonisolated public func
```

```
accentColor(_ accentColor: Color?) ->
some View

}

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

    /// Plays the given keyframes when
    the given trigger value changes, updating
        /// the view using the modifiers you
    apply in `body`.
    ///
    /// Note that the `content` closure
    will be updated on every frame while
        /// animating, so avoid performing
    any expensive operations directly within
        /// `content`.
    ///
    /// If the trigger value changes
    while animating, the `keyframes` closure
        /// will be called with the current
    interpolated value, and the keyframes
        /// that you return define a new
    animation that replaces the old one. The
        /// previous velocity will be
    preserved, so cubic or spring keyframes
    will
        /// maintain continuity from the
    previous animation if they do not specify
        /// a custom initial velocity.
    ///
}
```

```
    /// When a keyframe animation
finishes, the animator will remain at the
    /// end value, which becomes the
initial value for the next animation.
    ///
    /// - Parameters:
    ///   - initialValue: The initial
value that the keyframes will animate
    ///     from.
    ///   - trigger: A value to observe
for changes.
    ///   - content: A view builder
closure that takes two parameters. The
first
    ///     parameter is a proxy value
representing the modified view. The
    ///     second parameter is the
interpolated value generated by the
    ///     keyframes.
    ///   - keyframes: Keyframes defining
how the value changes over time. The
    ///     current value of the animator
is the single argument, which is
    ///     equal to `initialValue` when
the view first appears, then is equal
    ///     to the end value of the
previous keyframe animation on subsequent
    ///     calls.
nonisolated public func
keyframeAnimator<Value>(initialValue:
Value, trigger: some Equatable,
@ViewBuilder content: @escaping @Sendable
(PlaceholderContentView<Self>, Value) ->
```

```
some View, @KeyframesBuilder<Value>
keyframes: @escaping (Value) -> some
Keyframes) -> some View
```

```
    /// Loops the given keyframes
    continuously, updating
        /// the view using the modifiers you
        apply in `body`.
        ///
        /// Note that the `content` closure
        will be updated on every frame while
            /// animating, so avoid performing
        any expensive operations directly within
            /// `content`.
        ///
        /// – Parameters:
        ///     – initialValue: The initial
        value that the keyframes will animate
            ///     from.
        ///     – repeating: Whether the
        keyframes are currently repeating. If
        false,
            ///     the value at the beginning of
        the keyframe timeline will be
            ///     provided to the content
        closure.
        ///     – content: A view builder
        closure that takes two parameters. The
        first
            ///     parameter is a proxy value
        representing the modified view. The
            ///     second parameter is the
```

```
interpolated value generated by the
    ///      keyframes.
    /// - keyframes: Keyframes defining
how the value changes over time. The
    ///      current value of the animator
is the single argument, which is
    ///      equal to `initialValue` when
the view first appears, then is equal
    ///      to the end value of the
previous keyframe animation on subsequent
    ///      calls.

nonisolated public func
keyframeAnimator<Value>(initialValue:
Value, repeating: Bool = true,
@ViewBuilder content: @escaping @Sendable
(PlaceholderContentView<Self>, Value) ->
some View, @KeyframesBuilder<Value>
keyframes: @escaping (Value) -> some
Keyframes) -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Adds a different padding amount
to each edge of this view.
    ///
    /// Use this modifier to add a
different amount of padding on each edge
    /// of a view:
    ///
```

```
    ///      VStack {
    ///          Text("Text padded by
different amounts on each edge.")
    ///              .padding(EdgeInsets(t
op: 10, leading: 20, bottom: 40,
trailing: 0))
    ///                  .border(.gray)
    ///          Text("Unpadded text for
comparison.")
    ///                  .border(.yellow)
    ///      }
    ///
    ///      /// The order in which you apply
modifiers matters. The example above
    /// applies the padding before
applying the border to ensure that the
    /// border encompasses the padded
region:
    ///
    /// ! [A screenshot of two text
strings arranged vertically, each
surrounded
    /// by a border, with a small space
between the two borders.
    /// The first string says Text padded
by different amounts on each edge.
    /// Its border is gray, and there are
different amounts of space between
    /// the string and its border on each
edge: 40 points on the bottom, 10
    /// points on the top, 20 points on
the leading edge, and no space on
    /// the trailing edge.
```

```
    /// The second string says Unpadded  
text for comparison.  
    /// Its border is yellow, and there's  
no space between the string  
    /// and its border.] (View-padding-3-  
iOS)  
    ///  
    /// To pad a view on specific edges  
with equal padding for all padded  
    /// edges, use  
``View/padding(_:_:)``. To pad all edges  
of a view  
    /// equally, use  
``View/padding(_:)``.  
    ///  
    /// - Parameter insets: An  
``EdgeInsets`` instance that contains  
    /// padding amounts for each edge.  
    ///  
    /// - Returns: A view that's padded  
by different amounts on each edge.  
@inlinable nonisolated public func  
padding(_ insets: EdgeInsets) -> some  
View
```

```
    /// Adds an equal padding amount to  
specific edges of this view.  
    ///  
    /// Use this modifier to add a  
specified amount of padding to one or  
more  
    /// edges of the view. Indicate the
```

```
edges to pad by naming either a single
    /// value from ``Edge/Set``, or by
specifying an
    ///
<doc://com.apple.documentation/documentation/Swift/OptionSet>
    /// that contains edge values:
    ///
    ///     VStack {
    ///         Text("Text padded by 20
points on the bottom and trailing
edges.")
    ///             .padding([.bottom, .t
railing], 20)
    ///             .border(.gray)
    ///         Text("Unpadded text for
comparison.")
    ///             .border(.yellow)
    ///     }
    ///
    /// The order in which you apply
modifiers matters. The example above
    /// applies the padding before
applying the border to ensure that the
    /// border encompasses the padded
region:
    ///
    /// ! [A screenshot of two text
strings arranged vertically, each
surrounded
    /// by a border, with a small space
between the two borders.
    /// The first string says Text padded
```

```
by 20 points
    /// on the bottom and trailing edges.
    /// Its border is gray, and there are
20 points of space between the bottom
    /// and trailing edges of the string
and its border.
    /// There's no space between the
string and the border on the other edges.
    /// The second string says Unpadded
text for comparison.
    /// Its border is yellow, and there's
no space between the string
    /// and its border.])(View-padding-2-
iOS)
    ///
    /// You can omit either or both of
the parameters. If you omit the `length`,
    /// SwiftUI uses a default amount of
padding. If you
    /// omit the `edges`, SwiftUI applies
the padding to all edges. Omit both
    /// to add a default padding all the
way around a view. SwiftUI chooses a
    /// default amount of padding that's
appropriate for the platform and
    /// the presentation context.
    ///
    ///     VStack {
    ///         Text("Text with default
padding.")
    ///             .padding()
    ///             .border(.gray)
    ///         Text("Unpadded text for
```

```
comparison.")

    /**
     * border(.yellow)
     */

    /**
     * The example above looks like this
     * in iOS under typical conditions:
     */
    /**
     * ! [A screenshot of two text
     * strings arranged vertically, each
     * surrounded
     * by a border, with a small space
     * between the two borders.
     * The first string says Text with
     * default padding.
     * Its border is gray, and there is
     * padding on all sides
     * between the border and the string
     * it encloses in an amount that's
     * similar to the height of the
     * text.
     * The second string says Unpadded
     * text for comparison.
     * Its border is yellow, and there's
     * no space between the string
     * and its border.] (View-padding-2a-
     * iOS)
     */
    /**
     * To control the amount of padding
     * independently for each edge, use
     * ``View/padding(_:) -6pgqq``. To
     * pad all outside edges of a view by a
     * specified amount, use
     * ``View/padding(_:) -68shk``.

```

```
///  
/// - Parameters:  
///   - edges: The set of edges to  
pad for this view. The default  
///     is ``Edge/Set/all``.  
///   - length: An amount, given in  
points, to pad this view on the  
///     specified edges. If you set  
the value to `nil`, SwiftUI uses  
///     a platform-specific default  
amount. The default value of this  
///     parameter is `nil`.  
///  
/// - Returns: A view that's padded  
by the specified amount on the  
///     specified edges.  
@inlinable nonisolated public func  
padding(_ edges: Edge.Set = .all, _  
length: CGFloat? = nil) -> some View
```

```
/// Adds a specific padding amount to  
each edge of this view.  
///  
/// Use this modifier to add padding  
all the way around a view.  
///  
///     VStack {  
///         Text("Text padded by 10  
points on each edge.")  
///             .padding(10)  
///             .border(.gray)  
///         Text("Unpadded text for
```

```
comparison.")

    /**
     * border(.yellow)
     */

    /**
     * The order in which you apply
     * modifiers matters. The example above
     * applies the padding before
     * applying the border to ensure that the
     * border encompasses the padded
     * region:
     */
    /**
     * ! [A screenshot of two text
     * strings arranged vertically, each
     * surrounded
     * by a border, with a small space
     * between the two borders.
     * The first string says Text padded
     * by 10 points on each edge.
     * Its border is gray, and there are
     * 10 points of space on all sides
     * between the string and its
     * border.
     * The second string says Unpadded
     * text for comparison.
     * Its border is yellow, and there's
     * no space between the string
     * and its border.] (View-padding-1-
     * iOS)
     */
    /**
     * To independently control the
     * amount of padding for each edge, use
     * ``View/padding(_:) -6pgqq``. To
     * pad a select set of edges by the
```

```
    /// same amount, use
``View/padding(_:_:)``.
    ///
    /// - Parameter length: The amount,
given in points, to pad this view on all
    ///   edges.
    ///
    /// - Returns: A view that's padded
by the amount you specify.
@inlinable nonisolated public func
padding(_ length: CGFloat) -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Sets the blend mode for
compositing this view with overlapping
views.
    ///
    /// Use `blendMode(_:)` to combine
overlapping views and use a different
    /// visual effect to produce the
result. The ``BlendMode`` enumeration
    /// defines many possible effects.
    ///
    /// In the example below, the two
overlapping rectangles have a
    /// ``BlendMode/colorBurn`` effect
applied, which effectively removes the
    /// non-overlapping portion of the
```

```
second image:
```

```
    /**
     * HStack {
     *     Color.yellow.frame(width:
     * 50, height: 50, alignment: .center)
     *
     *     Color.red.frame(width:
     * 50, height: 50, alignment: .center)
     *         .rotationEffect(.degr
     * ees(45))
     *
     *         .padding(-20)
     *         .blendMode(.colorBurn
     *)
     */
     /**
     * ! [Two overlapping rectangles
     showing the effect of the blend mode view
     modifier applying the colorBurn
     effect.] (SwiftUI-blendMode.png)
     */
     /**
     * - Parameter blendMode: The
     ``BlendMode`` for compositing this view.
     */
     /**
     * - Returns: A view that applies
     `blendMode` to this view.
     @inlinable nonisolated public func
     blendMode(_ blendMode: BlendMode) -> some
     View
     }

     @available(iOS 15.0, macOS 12.0, tvOS
     15.0, watchOS 8.0, *)
```

```
extension View {

    /// Marks the view as containing
    sensitive, private user data.
    ///
    /// SwiftUI redacts views marked with
    this modifier when you apply the
    /// ``RedactionReasons/privacy``
    redaction reason.
    ///
    /// struct BankAccountView: View
{
    ///
    var body: some View {
        ///
        VStack {
            ///
            Text("Account #")
            ///
            ///
            Text(accountNumber)
            ///
            .font(.headline)
            ///
            .privacySensitive() // Hide only the account number.
            ///
            }
            ///
            }
            ///
            }

    nonisolated public func
    privacySensitive(_ sensitive: Bool =
    true) -> some View
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
```

```
extension View {
```

```
    /// Sets the contrast and separation  
    between similar colors in this view.
```

```
    ///
```

```
    /// Apply contrast to a view to  
    increase or decrease the separation  
    between
```

```
    /// similar colors in the view.
```

```
    ///
```

```
    /// In the example below, the  
    `contrast(_:)` modifier is applied to a  
    set of
```

```
    /// red squares each containing a  
    contrasting green inner circle. At each
```

```
    /// step in the loop, the  
    `contrast(_:)` modifier changes the  
    contrast of
```

```
    /// the circle/square view in 20%  
    increments. This ranges from -20%  
    contrast
```

```
    /// (yielding inverted colors –  
    turning the red square to pale-green and the
```

```
    /// green circle to mauve), to  
    neutral-gray at 0%, to 100% contrast
```

```
    /// (bright-red square / bright-green  
    circle). Applying negative contrast
```

```
    /// values, as shown in the -20%  
    square, will apply contrast in addition  
    to
```

```
    /// inverting colors.
```

```
    ///
```

```
/// struct CircleView: View {
///     var body: some View {
///         Circle()
///             .fill(Color.green)
///     }
///         .frame(width: 25,
height: 25, alignment: .center)
///     }
/// }
/// struct Contrast: View {
///     var body: some View {
///         HStack {
///             ForEach(-1..<6) {
///                 Color.red.frame(width: 50, height: 50,
alignment: .center)
///                     .overlay(
CircleView(), alignment: .center)
///                     .contrast
(Double($0) * 0.2)
///                     .overlay(
Text("\(Double($0) * 0.2 * 100,
specifier: "%.0f")%")
/// .font(.callout),
/// alignment: .bottom)
/// .border(C
olor.gray)
/// }
/// }
/// }
```

```
    /**
     */
    /**
     * ! [Demonstration of the effect of
     * contrast on a view applying contrast
     * values from -20% to 100%
     * contrast.] (SwiftUI-View-contrast.png)
     */
    /**
     * - Parameter amount: The intensity
     * of color contrast to apply. negative
     * values invert colors in
     * addition to applying contrast.
     */
    /**
     * - Returns: A view that applies
     * color contrast to this view.
     @inlinable nonisolated public func
     contrast(_ amount: Double) -> some View
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension View {

    /**
     * Defines a group of views with
     * synchronized geometry using an
     * identifier and namespace that you
     * provide.
     */
    /**
     * This method sets the geometry of
     * each view in the group from the
     * inserted view with `isSource =
     * true` (known as the "source" view),
     * updating the values marked by
```

```
`properties`.  
    ///  
    /// If inserting a view in the same  
    transaction that another view  
    /// with the same key is removed, the  
    system will interpolate their  
    /// frame rectangles in window space  
    to make it appear that there  
    /// is a single view moving from its  
    old position to its new  
    /// position. The usual transition  
    mechanisms define how each of  
    /// the two views is rendered during  
    the transition (e.g. fade  
    /// in/out, scale, etc), the  
`matchedGeometryEffect()` modifier  
    /// only arranges for the geometry of  
the views to be linked, not  
    /// their rendering.  
    ///  
    /// If the number of currently-  
inserted views in the group with  
    /// `isSource = true` is not exactly  
one results are undefined, due  
    /// to it not being clear which is  
the source view.  
    ///  
    /// - Parameters:  
    ///   - id: The identifier, often  
derived from the identifier of  
    ///       the data being displayed by  
the view.  
    ///   - namespace: The namespace in
```

which defines the `id`. New  
    /// namespaces are created by  
    adding an `@Namespace` variable  
    /// to a ``View`` type and  
    reading its value in the view's body  
    /// method.  
    /// - properties: The properties to  
    copy from the source view.  
    /// - anchor: The relative location  
    in the view used to produce  
    /// its shared position value.  
    /// - isSource: True if the view  
    should be used as the source of  
    /// geometry for other views in  
    the group.  
    ///  
    /// - Returns: A new view that  
    defines an entry in the global  
    /// database of views synchronizing  
    their geometry.  
    ///  
    **@inlinable nonisolated public func**  
**matchedGeometryEffect<ID>(id: ID, in**  
**namespace: Namespace.ID, properties:**  
**MatchedGeometryProperties = .frame,**  
**anchor: UnitPoint = .center, isSource:**  
**Bool = true) -> some View where ID : Hashable**  
}  
  
**@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, \*)**

```
extension View {

    /// Sets the environment value of the
    /// specified key path to the given value.
    ///
    /// Use this modifier to set one of
    /// the writable properties of the
    /// ``EnvironmentValues`` structure,
    /// including custom values that you
    /// create. For example, you can set
    /// the value associated with the
    ///
    /// ``EnvironmentValues/truncationMode`` key:
    ///
    ///     MyView()
    ///     .environment(\.truncation
    /// Mode, .head)
    ///
    /// You then read the value inside
    /// `MyView` or one of its descendants
    /// using the ``Environment``
    /// property wrapper:
    ///
    ///     struct MyView: View {
    ///
    /// @Environment(\.truncationMode) var
    /// truncationMode: Text.TruncationMode
    ///
    ///     var body: some View { ...
    /// }
    ///
    ///     }
    ///
    /// SwiftUI provides dedicated view
```

modifiers for setting most  
    /// environment values, like the  
``View/truncationMode(\_:)``  
    /// modifier which sets the  
``EnvironmentValues/truncationMode``  
value:  
    ///  
    ///     MyView()  
    ///                 .truncationMode(.head)  
    ///  
    /// Prefer the dedicated modifier  
when available, and offer your own when  
    /// defining custom environment  
values, as described in  
    /// ``Entry()``.  
    ///  
    /// This modifier affects the given  
view,  
    /// as well as that view's descendant  
views. It has no effect  
    /// outside the view hierarchy on  
which you call it.  
    ///  
    /// - Parameters:  
    ///     - keyPath: A key path that  
indicates the property of the  
    ///        ``EnvironmentValues``  
structure to update.  
    ///     - value: The new value to set  
for the item specified by `keyPath`.  
    ///  
    /// - Returns: A view that has the  
given value set in its environment.

```
    @inlinable nonisolated public func
environment<V>(_ keyPath:
WritableKeyPath<EnvironmentValues, V>, _  
value: V) -> some View  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Layers a secondary view in front  
of this view.  
    ///  
    /// When you apply an overlay to a  
view, the original view continues to  
    /// provide the layout  
characteristics for the resulting view.  
In the  
    /// following example, the heart  
image is shown overlaid in front of, and  
    /// aligned to the bottom of the  
folder image.  
    ///  
    ///     Image(systemName: "folder")  
    ///             .font(.system(size: 55,  
weight: .thin))  
    ///             .overlay(Text("♥"),  
alignment: .bottom)  
    ///  
    /// ! [View showing placement of a  
heart overlaid onto a folder  
    /// icon.] (View-overlay-1)
```

```
///  
/// - Parameters:  
///   - overlay: The view to layer in  
front of this view.  
///   - alignment: The alignment for  
`overlay` in relation to this view.  
///  
/// - Returns: A view that layers  
`overlay` in front of the view.  
@available(iOS, introduced: 13.0,  
deprecated: 100000.0, message: "Use  
`overlay(alignment:content:)` instead.")  
@available(macOS, introduced: 10.15,  
deprecated: 100000.0, message: "Use  
`overlay(alignment:content:)` instead.")  
@available(tvOS, introduced: 13.0,  
deprecated: 100000.0, message: "Use  
`overlay(alignment:content:)` instead.")  
@available(watchOS, introduced: 6.0,  
deprecated: 100000.0, message: "Use  
`overlay(alignment:content:)` instead.")  
@available(visionOS, introduced: 1.0,  
deprecated: 100000.0, message: "Use  
`overlay(alignment:content:)` instead.")  
@inlinable nonisolated public func  
overlay<Overlay>(_ overlay: Overlay,  
alignment: Alignment = .center) -> some  
View where Overlay : View
```

```
/// Adds a border to this view with  
the specified style and width.  
///
```

```
    /// Use this modifier to draw a
    border of a specified width around the
    /// view's frame. By default, the
    border appears inside the bounds of this
    /// view. For example, you can add a
    four-point wide border covers the text:
    ///
    ///     Text("Purple border inside
the view bounds.")
    ///         .border(Color.purple,
width: 4)
    ///
    /// ! [A screenshot showing the text
Purple border inside the view bounds.
    /// The text is surrounded by a
purple border that outlines the text,
    /// but isn't quite big enough and
encroaches on the text.] (View-border-1)
    ///
    /// To place a border around the
outside of this view, apply padding of
the
    /// same width before adding the
border:
    ///
    ///     Text("Purple border outside
the view bounds.")
    ///         .padding(4)
    ///         .border(Color.purple,
width: 4)
    ///
    /// ! [A screenshot showing the text
Purple border outside the view bounds.
```

```
    /// The text is surrounded by a
    purple border that outlines the text
    /// without touching the text.] (View-
border-2)
    /**
     /**
     /**
     - Parameters:
     /**
     - content: A value that
conforms to the ``ShapeStyle`` protocol,
     /**
     like a ``Color`` or
``HierarchicalShapeStyle``, that SwiftUI
     /**
     uses to fill the border.
     /**
     - width: The thickness of the
border. The default is 1 pixel.
    /**
     /**
     /**
     - Returns: A view that adds a
border with the specified style and width
     /**
     to this view.
@inlinable nonisolated public func
border<S>(_ content: S, width: CGFloat =
1) -> some View where S : ShapeStyle
}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension View {

    /**
     /**
     /**
     - Layers the views that you specify
in front of this view.
    /**
     /**
     /**
     - Use this modifier to place one or
more views in front of another view.
     /**
     /**
     /**
     - For example, you can place a
```

```
group of stars on a ``RoundedRectangle``:  
    ///  
    ///  
    RoundedRectangle(cornerRadius: 8)  
        /// .frame(width: 200,  
height: 100)  
        /// .overlay(alignment: .topL  
eading) { Star(color: .red) }  
        /// .overlay(alignment: .topT  
railing) { Star(color: .yellow) }  
        /// .overlay(alignment: .bott  
omLeading) { Star(color: .green) }  
        /// .overlay(alignment: .bott  
omTrailing) { Star(color: .blue) }  
    ///  
    /// The example above assumes that  
you've defined a `Star` view with a  
    /// parameterized color:  
    ///  
    ///     struct Star: View {  
    ///         var color = Color.yellow  
    ///  
    ///         var body: some View {  
    ///             Image(systemName:  
"star.fill")  
    ///                 .foregroundStyle(  
color)  
    ///             }  
    ///         }  
    ///  
    /// By setting different `alignment`  
values for each modifier, you make the  
    /// stars appear in different places
```

on the rectangle:

```
///
/// ! [A screenshot of a rounded
rectangle with a star in each corner. The
/// star in the upper-left is red;
the start in the upper-right is yellow;
/// the star in the lower-left is
green; the star the lower-right is
/// blue.] (View-overlay-2)
///
/// If you specify more than one view
in the `content` closure, the modifier
/// collects all of the views in the
closure into an implicit ``ZStack``,
/// taking them in order from back to
front. For example, you can place a
/// star and a ``Circle`` on a field
of ``ShapeStyle/blue``:
///
///     Color.blue
///         .frame(width: 200,
height: 200)
///             .overlay {
///                 Circle()
///                     .frame(width:
100, height: 100)
///                         Star()
///                     }
///
/// Both the overlay modifier and the
implicit ``ZStack`` composed from the
/// overlay content --- the circle
and the star --- use a default
```

```
    /// ``Alignment/center`` alignment.  
The star appears centered on the circle,  
    /// and both appear as a composite  
view centered in front of the square:  
    ///  
    /// ! [A screenshot of a star centered  
on a circle, which is  
    /// centered on a square.] (View-  
overlay-3)  
    ///  
    /// If you specify an alignment for  
the overlay, it applies to the implicit  
    /// stack rather than to the  
individual views in the closure. You can  
see  
    /// this if you add the  
``Alignment/bottom`` alignment:  
    ///  
    ///     Color.blue  
    ///         .frame(width: 200,  
height: 200)  
    ///             .overlay(alignment: .bott  
om) {  
    ///                 Circle()  
    ///                     .frame(width:  
100, height: 100)  
    ///                     Star()  
    ///                 }  
    ///  
    /// The circle and the star move down  
as a unit to align the stack's bottom  
    /// edge with the bottom edge of the  
square, while the star remains
```

```
    /// centered on the circle:  
    ///  
    /// ! [A screenshot of a star centered  
on a circle, which is on a square.  
    /// The circle's bottom edge is  
aligned with the square's bottom  
    /// edge.] (View-overlay-3a)  
    ///  
    /// To control the placement of  
individual items inside the `content`  
    /// closure, either use a different  
overlay modifier for each item, as the  
    /// earlier example of stars in the  
corners of a rectangle demonstrates, or  
    /// add an explicit ``ZStack`` inside  
the content closure with its own  
    /// alignment:  
    ///  
    ///     Color.blue  
    ///         .frame(width: 200,  
height: 200)  
    ///             .overlay(alignment: .bott  
om) {  
    ///  
ZStack(alignment: .bottom) {  
    ///                 Circle()  
    ///                     .frame(width:  
100, height: 100)  
    ///                         Star()  
    ///                     }  
    ///                 }  
    ///  
    /// The stack alignment ensures that
```

the star's bottom edge aligns with the  
    /// circle's, while the overlay  
aligns the composite view with the  
square:

```
    ///  
    /// ! [A screenshot of a star, a  
circle, and a square with all their  
    /// bottom edges aligned.] (View-  
overlay-4)  
    ///  
    /// You can achieve layering without  
an overlay modifier by putting both the  
    /// modified view and the overlay  
content into a ``ZStack``. This can  
    /// produce a simpler view hierarchy,  
but changes the layout priority that  
    /// SwiftUI applies to the views. Use  
the overlay modifier when you want the  
    /// modified view to dominate the  
layout.  
    ///  
    /// If you want to specify a  
``ShapeStyle`` like a ``Color`` or a  
    /// ``Material`` as the overlay, use  
    ///  
``View/overlay(_:_ignoresSafeAreaEdges:)``  
instead. To specify a  
    /// ``Shape``, use  
``View/overlay(_:_in:_fillStyle:)``.  
    ///  
    /// - Parameters:  
    ///   - alignment: The alignment that  
the modifier uses to position the
```

```
    ///      implicit ``ZStack`` that
groups the foreground views. The default
    ///      is ``Alignment/center``.
    /// - content: A ``ViewBuilder``
that you use to declare the views to
    ///      draw in front of this view,
stacked in the order that you list them.
    ///      The last view that you list
appears at the front of the stack.
    ///
    /// - Returns: A view that uses the
specified content as a foreground.
@inlinable nonisolated public func
overlay<V>(alignment: Alignment
= .center, @ViewBuilder content: () -> V)
-> some View where V : View
```

```
    /// Layers the specified style in
front of this view.
    ///
    /// Use this modifier to layer a type
that conforms to the ``ShapeStyle``
    /// protocol, like a ``Color``,
``Material``, or
``HierarchicalShapeStyle``,
    /// in front of a view. For example,
you can overlay the
    /// ``ShapeStyle/ultraThinMaterial``
over a ``Circle``:
    ///
    ///     struct CoveredCircle: View {
    ///         var body: some View {
```

```
    /**
     * Circle()
     *   .frame(width:
300, height: 200)
     *   .overlay(.ultraTh
inMaterial)
    /**
     */
    /**
     */
    /**
     * SwiftUI anchors the style to the
view's bounds. For the example above,
     * the overlay fills the entirety of
the circle's frame (which happens
     * to be wider than the circle is
tall):
    /**
     * ! [A screenshot of a circle
showing through a rectangle that imposes
     * a blurring effect.] (View-
overlay-5)
    /**
     * SwiftUI also limits the style's
extent to the view's
     * container-relative shape. You can
see this effect if you constrain the
     * `CoveredCircle` view with a
``View/containerShape(_:)`` modifier:
    /**
     * CoveredCircle()
     *   .containerShape(RoundedRe
ctangle(cornerRadius: 30))
    /**
     * The overlay takes on the
specified container shape:
```

```
///  
/// ! [A screenshot of a circle  
showing through a rounded rectangle that  
/// imposes a blurring effect.] (View-  
overlay-6)  
///  
/// By default, the overlay ignores  
safe area insets on all edges, but you  
/// can provide a specific set of  
edges to ignore, or an empty set to  
/// respect safe area insets on all  
edges:  
///  
///     Rectangle()  
///         .overlay(  
///             .secondary,  
///             ignoresSafeAreaEdges:  
[])) // Ignore no safe area insets.  
///  
/// If you want to specify a ``View``  
or a stack of views as the overlay  
/// rather than a style, use  
``View/overlay(alignment:content:)``  
instead.  
/// If you want to specify a  
``Shape``, use  
///  
``View/overlay(_:in:fillStyle:)``.  
///  
/// - Parameters:  
///     - style: An instance of a type  
that conforms to ``ShapeStyle`` that  
///     SwiftUI layers in front of
```

the modified view.

    /// - edges: The set of edges for which to ignore safe area insets

    /// when adding the overlay. The default value is ``Edge/Set/all``.

    /// Specify an empty set to respect safe area insets on all edges.

    ///

    /// - Returns: A view with the specified style drawn in front of it.

```
@inlinable nonisolated public func
overlay<S>(_ style: S,
ignoresSafeAreaEdges edges: Edge.Set
= .all) -> some View where S : ShapeStyle
```

    /// Layers a shape that you specify in front of this view.

    ///

    /// Use this modifier to layer a type that conforms to the ``Shape``

    /// protocol --- like a ``Rectangle``, ``Circle``, or ``Capsule`` --- in

    /// front of a view. Specify a ``ShapeStyle`` that's used to fill the shape.

    /// For example, you can overlay the outline of one rectangle in front of

    /// another:

    ///

    ///     Rectangle()

    ///                 .frame(width: 200,

```
height: 100)
    /// .overlay(.teal, in:
Rectangle().inset(by:
10).stroke(lineWidth: 5))
    ///
    /// The example above uses the
``InsettableShape/inset(by:)`` method to
    /// slightly reduce the size of the
overlaid rectangle, and the
    /// ``Shape/stroke(lineWidth:)``
method to fill only the shape's outline.
    /// This creates an inset border:
    ///
    /// ![A screenshot of a rectangle
with a teal border that's
    /// inset from the edge.](View-
overlay-7)
    ///
    /// This modifier is a convenience
method for layering a shape over a view.
    /// To handle the more general case
of overlaying a ``View`` --- or a stack
    /// of views --- with control over
the position, use
    ///
``View/overlay(alignment:content:)``
instead. To cover a view with a
    /// ``ShapeStyle``, use
``View/overlay(_:ignoresSafeAreaEdges:)``
-
    ///
    /// - Parameters:
    ///   - style: A ``ShapeStyle`` that
```

```
SwiftUI uses to the fill the shape
    ///      that you specify.
    /// - shape: An instance of a type
that conforms to ``Shape`` that
    ///      SwiftUI draws in front of the
view.
    /// - fillStyle: The ``FillStyle``
to use when drawing the shape.
    ///      The default style uses the
nonzero winding number rule and
    ///      antialiasing.
    ///
    /// - Returns: A view with the
specified shape drawn in front of it.
@inlinable nonisolated public func
overlay<S, T>(_ style: S, in shape: T,
fillStyle: FillStyle = FillStyle()) ->
some View where S : ShapeStyle, T : Shape

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Applies a Gaussian blur to this
view.
    ///
    /// Use `blur(radius:opaque:)` to
apply a gaussian blur effect to the
    /// rendering of this view.
    ///
    /// The example below shows two
```

```
``Text`` views, the first with no blur
    /// effects, the second with
`blur(radius:opaque:)` applied with the
    /// `radius` set to `2`. The larger
the radius, the more diffuse the
    /// effect.

    ///
    /**
     struct Blur: View {
     var body: some View {
     VStack {
     Text("This is
some text.")
     /**
     .padding()
     /**
     Text("This is
some blurry text.")
     /**
     .blur(radius:
2.0)
     /**
     }
     /**
     }
     /**
     */
     /**
     ! [A screenshot showing the effect
of applying gaussian blur effect to
     /// the rendering of a view.]
(SwiftUI-View-blurRadius.png)

    /**
    /// - Parameters:
    ///   - radius: The radial size of
the blur. A blur is more diffuse when its
    ///   radius is large.
    ///   - opaque: A Boolean value that
indicates whether the blur renderer
    ///   permits transparency in the
```

```
blur output. Set to `true` to create an
    ///      opaque blur, or set to
`false` to permit transparency.
    @inlinable nonisolated public func
blur(radius: CGFloat, opaque: Bool =
false) -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Inverts the colors in this view.
    ///
    /// The `colorInvert()` modifier
    inverts all of the colors in a view so
    that
        /// each color displays as its
    complementary color. For example, blue
        /// converts to yellow, and white
    converts to black.
    ///
    /// In the example below, two red
    squares each have an interior green
        /// circle. The inverted square shows
    the effect of the square's colors:
        /// complimentary colors for red and
    green – teal and purple.
    ///
    ///     struct InnerCircleView: View
{
    ///             var body: some View {
```

```
    /**
     * Circle()
     *   .fill(Color.green)
     */
    /**
     *   .frame(width: 40,
     * height: 40, alignment: .center)
     */
    /**
     */
    /**
     */
    /**
     * struct ColorInvert: View {
     *   var body: some View {
     *     HStack {
     *
     * Color.red.frame(width: 100, height: 100,
     * alignment: .center)
     *   /**
     *     .overlay(Inne
     * rCircleView(), alignment: .center)
     *   /**
     *     .overlay(Text
     * ("Normal"))
     *   /**
     *   .font(.callout),
     *   /**
     * alignment: .bottom)
     *   /**
     *     .border(Color
     * .gray)
     *   /**
     *   /**
     *     Spacer()
     *   /**
     *   /**
     *   /**
     *
     * Color.red.frame(width: 100, height: 100,
     * alignment: .center)
     *   /**
     *     .overlay(Inne
     * rCircleView(), alignment: .center)
     *   /**
     *     .colorInvert(
```

```
)  
    /// .overlay(Text  
("Inverted")  
    /// .font(.callout),  
    /// alignment: .bottom)  
    /// .border(Color  
.gray)  
    /// }  
    /// .padding(50)  
    /// }  
    ///  
    /// ! [Two red squares with centered  
green circles with one showing the  
/// effect of color inversion, which  
yields teal and purple replacing the  
/// red and green colors.] (SwiftUI-  
View-colorInvert.png)  
    ///  
    /// - Returns: A view that inverts  
its colors.  
    @inlinable nonisolated public func  
colorInvert() -> some View  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Adds a grayscale effect to this
```

view.

```
///  
/// A grayscale effect reduces the  
intensity of colors in this view.  
///  
/// The example below shows a series  
of red squares with their grayscale  
/// effect increasing from 0  
(reddest) to 99% (fully desaturated) in  
/// approximate 20% increments:  
///  
///     struct Saturation: View {  
///         var body: some View {  
///             HStack {  
///                 ForEach(0..<6) {  
///                     Color.red.frame(width: 60, height: 60,  
/// alignment: .center)  
///                         .grayscale(Double($0) * 0.1999)  
///                         .overlay(  
///                             Text("\u00d7(Double($0) * 0.1999 * 100,  
specifier: "%.4f")%"),  
///                             alignment: .bottom)  
///                         .border(C  
olor.gray)  
///                     }  
///                 }  
///             }  
///         }  
///     }  
/// ! [Rendering showing the effects
```

```
of grayscale adjustments in
    /// approximately 20% increments from
    fully-red at 0% to fully desaturated
    /// at 99%.](SwiftUI-View-
grayscale.png)
    /**
     /// - Parameter amount: The intensity
     of grayscale to apply from 0.0 to less
     /// than 1.0. Values closer to 0.0
     are more colorful, and values closer to
     /// 1.0 are less colorful.
     /**
     /// - Returns: A view that adds a
     grayscale effect to this view.
    @inlinable nonisolated public func
grayscale(_ amount: Double) -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Sets the transparency of this
view.
    /**
     /// Apply opacity to reveal views
that are behind another view or to
    /// de-emphasize a view.
    /**
     /// When applying the `opacity(_:)` modifier
to a view that has already had
    /// its opacity transformed, the
```

modifier multiplies the effect of the  
    /// underlying opacity  
transformation.

```
///
/// The example below shows yellow
and red rectangles configured to overlap.
/// The top yellow rectangle has its
opacity set to 50%, allowing the
/// occluded portion of the bottom
rectangle to be visible:
///
/// struct Opacity: View {
///     var body: some View {
///         VStack {
///             Color.yellow.frame(width: 100, height:
100, alignment: .center)
///                 .zIndex(1)
///                 .opacity(0.5)
///             Color.red.frame(width: 100, height: 100,
alignment: .center)
///                 .padding(-40)
///             }
///         }
///     }
/// ! [Two overlaid rectangles, where
the topmost has its opacity set to 50%,
/// which allows the occluded portion
of the bottom rectangle to be
/// visible.] (SwiftUI-View-
```

```
opacity.png)
    /**
     * - Parameter opacity: A value
     * between 0 (fully transparent) and 1
     * (fully
     *   /// opaque).
     *
     * - Returns: A view that sets the
     * transparency of this view.
     */
    @inlinable nonisolated public func
    opacity(_ opacity: Double) -> some View
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /**
     * Adds a shadow to this view.
     *
     * Use this modifier to add a shadow
     * of a specified color behind a view.
     * You can offset the shadow from
     * its view independently in the horizontal
     * and vertical dimensions using the
     * `x` and `y` parameters. You can also
     * blur the edges of the shadow
     * using the `radius` parameter. Use a
     * radius of zero to create a sharp
     * shadow. Larger radius values produce
     * softer shadows.
     *
     * The example below creates a grid

```





edges. Because the shadow of the box in the

/// upper left is both completely sharp and directly below the box, it isn't

/// visible.] (View-shadow-1-iOS)

///

/// The example above uses ``Color/primary`` as the color to make the

/// shadow easy to see for the purpose of illustration. In practice,

/// you might prefer something more subtle, like ``Color/gray-8j2b``.

/// If you don't specify a color, the method uses a semi-transparent

/// black.

///

/// - Parameters:

/// - color: The shadow's color.

/// - radius: A measure of how much to blur the shadow. Larger values

/// result in more blur.

/// - x: An amount to offset the shadow horizontally from the view.

/// - y: An amount to offset the shadow vertically from the view.

///

/// - Returns: A view that adds a shadow to this view.

**@inlinable nonisolated public func**  
**shadow(color: Color = Color(.sRGBLinear,**  
**white: 0, opacity: 0.33), radius:**

```
CGFloat, x: CGFloat = 0, y: CGFloat = 0)
-> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Composites this view's contents
    /// into an offscreen image before final
    /// display.
    ///
    /// The
    `drawingGroup(opaque:colorMode:)` modifier flattens a subtree of
    /// views into a single view before
    rendering it.
    ///
    /// In the example below, the
    contents of the view are composited to a
    /// single bitmap; the bitmap is then
    displayed in place of the view:
    ///
    ///     VStack {
    ///         ZStack {
    ///             Text("DrawingGroup")
    ///                 .foregroundColor(
    ///                     .black)
    ///                 .padding(20)
    ///                 .background(Color
    ///                     .red)
    ///             Text("DrawingGroup")
    ///         }
    ///     }
}
```

```
    ///         .blur(radius: 2)
    ///     }
    ///     .font(.largeTitle)
    ///     .compositingGroup()
    ///     .opacity(1.0)
    /// }
    /// .background(Color.white)
    /// .drawingGroup()

    /// > Note: Views backed by native
platform views may not render into the
    /// image. Instead, they log a
warning and display a placeholder image
to
    /// highlight the error.

    ///
    /// ! [A screenshot showing the
effects on several stacks configured as a
    /// drawing group.] (SwiftUI-View-
drawingGroup.png)
    ///
    /// - Parameters:
    ///   - opaque: A Boolean value that
indicates whether the image is opaque.
    ///   The default is `false`; if
set to `true`, the alpha channel of the
    ///   image must be `1`.
    ///   - colorMode: One of the working
color space and storage formats
    ///   defined in
``ColorRenderingMode``. The default is
    ///
``ColorRenderingMode/nonLinear``.
```

```
    /**
     * - Returns: A view that composites
     * this view's contents into an offscreen
     * image before display.
     */
    nonisolated public func
    drawingGroup(opaque: Bool = false,
    colorMode: ColorRenderingMode
    = .nonLinear) -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /**
     * Brightens this view by the
     * specified amount.
     */
    /**
     * Use `brightness(_:)` to brighten
     * the intensity of the colors in a view.
     * The example below shows a series
     * of red squares, with their brightness
     * increasing from 0 (fully red) to
     * 100% (white) in 20% increments.
     */
    /**
     * struct Brightness: View {
     *     var body: some View {
     *         HStack {
     *             ForEach(0..<6) {
     *                 Color.red.frame(width: 60, height: 60,
     * alignment: .center)
     *                     .brightne
```



```
    /// Isolates the geometry (e.g.  
position and size) of the view  
    /// from its parent view.  
    ///  
    /// By default SwiftUI views push  
position and size changes down  
    /// through the view hierarchy, so  
that only views that draw  
    /// something (known as leaf views)  
apply the current animation to  
    /// their frame rectangle. However in  
some cases this coalescing  
    /// behavior can give undesirable  
results; inserting a geometry  
    /// group can correct that. A group  
acts as a barrier between the  
    /// parent view and its subviews,  
forcing the position and size  
    /// values to be resolved and  
animated by the parent, before being  
    /// passed down to each subview.  
    ///  
    /// The example below shows one use  
of this function: ensuring that  
    /// the member views of each row in  
the stack apply (and animate  
    /// as) a single geometric transform  
from their ancestor view,  
    /// rather than letting the effects  
of the ancestor views be  
    /// applied separately to each leaf  
view. If the members of
```

```
    /// `ItemView` may be added and
    removed at different times the
    /// group ensures that they stay
    locked together as animations are
    /// applied.

    ///
    ///     VStack {
    ///         ForEach(items) { item in
    ///             ItemView(item: item)
    ///             .geometryGroup()
    ///         }
    ///     }

    /// Returns: a new view whose
    geometry is isolated from that of its
    /// parent view.
    nonisolated public func
    geometryGroup() -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Supplies an observable object to
    a view's hierarchy.
    ///
    /// Use this modifier to add an
    observable object to a view's
    environment.
    /// The object must conform to the
    ///
```

```
<doc://com.apple.documentation/documentation/Combine/ObservableObject>
    /// protocol.
    ///
    /// Adding an object to a view's
    environment makes the object available to
    /// subviews in the view's hierarchy.
    To retrieve the object in a subview,
    /// use the ``EnvironmentObject``
    property wrapper.
    ///
    /// > Note: If the observable object
    conforms to the
    ///
<doc://com.apple.documentation/documentation/Observation/Observable>
    /// protocol, use either
    ``View/environment(_:)`` or the
    /// ``View/environment(_:_:)``
    modifier to add the object to the view's
    /// environment.
    ///
    /// - Parameter object: The object to
    store and make available to
    /// the view's hierarchy.
    @inlinable nonisolated public func
environmentObject<T>(_ object: T) -> some
View where T : ObservableObject

}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
```

```
extension View {  
  
    /// Masks this view using the alpha  
    channel of the given view.  
    ///  
    /// Use `mask(_:)` when you want to  
    apply the alpha (opacity) value of  
    /// another view to the current view.  
    ///  
    /// This example shows an image  
    masked by rectangle with a 10% opacity:  
    ///  
    ///     Image(systemName:  
    "envelope.badge.fill")  
    ///         .foregroundColor(Color.bl  
ue)  
    ///         .font(.system(size: 128,  
    weight: .regular))  
    ///         .mask {  
    ///             Rectangle().opacity(0.1)  
    ///         }  
    ///  
    ///     ! [A screenshot of a view masked  
    by a rectangle with 10%  
    /// opacity.] (SwiftUI-View-mask.png)  
    ///  
    /// - Parameters:  
    ///     - alignment: The alignment  
    for `mask` in relation to this view.  
    ///     - mask: The view whose alpha  
    the rendering system applies to  
    ///         the specified view.
```

```
    @inlinable nonisolated public func
mask<Mask>(alignment: Alignment
= .center, @ViewBuilder _ mask: () ->
Mask) -> some View where Mask : View
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Masks this view using the alpha
    channel of the given view.
    ///
    /// Use `mask(_:)` when you want to
    apply the alpha (opacity) value of
    /// another view to the current view.
    ///
    /// This example shows an image
    masked by rectangle with a 10% opacity:
    ///
    ///     Image(systemName:
    "envelope.badge.fill")
    ///         .foregroundColor(Color.bl
    ue)
    ///         .font(.system(size: 128,
    weight: .regular))
    ///         .mask(Rectangle().opacity
    (0.1))
    ///
    /// ! [A screenshot of a view masked
    by a rectangle with 10%
    /// opacity.] (SwiftUI-View-mask.png)
```

```
    /**
     * - Parameter mask: The view whose
     * alpha the rendering system applies to
     * the specified view.
     @available(iOS, deprecated: 100000.0,
message: "Use overload where mask accepts
a @ViewBuilder instead.")
     @available(macOS, deprecated:
100000.0, message: "Use overload where
mask accepts a @ViewBuilder instead.")
     @available(tvOS, deprecated:
100000.0, message: "Use overload where
mask accepts a @ViewBuilder instead.")
     @available(watchOS, deprecated:
100000.0, message: "Use overload where
mask accepts a @ViewBuilder instead.")
     @available(visionOS, deprecated:
100000.0, message: "Use overload where
mask accepts a @ViewBuilder instead.")
     @inlinable nonisolated public func
mask<Mask>(_ mask: Mask) -> some View
where Mask : View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /**
     * Wraps this view in a compositing
     * group.
     /**
     * A compositing group makes
```

compositing effects in this view's ancestor

```
    /// views, such as opacity and the
    blend mode, take effect before this view
    /// is rendered.
    ///
    /// Use `compositingGroup()` to apply
    effects to a parent view before
    /// applying effects to this view.
    ///
    /// In the example below the
    `compositingGroup()` modifier separates
    the
    /// application of effects into
    stages. It applies the
    ``View/opacity(_:)``
    /// effect to the VStack before the
    `blur(radius:)` effect is applied to the
    /// views inside the enclosed
    ``ZStack``. This limits the scope of the
    /// opacity change to the outermost
    view.
    ///
    ///     VStack {
    ///         ZStack {
    ///
    Text("CompositingGroup")
        ///                         .foregroundColor(
    .black)
        ///                         .padding(20)
        ///                         .background(Color
    .red)
        ///
```

```
Text("CompositingGroup")
    /**
     * Returns: A view that wraps this view in a compositing group.
     */
    @inlinable nonisolated public func
    compositingGroup() -> some View
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension View {

    /**
     * - Parameters:
     *   - isActive: A Boolean value
     *     that indicates whether underline
     *       is added. The default value
     */
}
```

```
is `true`.  
    /// - pattern: The pattern of the  
line. The default value is `solid`.  
    /// - color: The color of the  
underline. If `color` is `nil`, the  
    ///     underline uses the default  
foreground color.  
    ///  
    /// - Returns: A view where text has  
a line running along its baseline.  
    @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
    nonisolated public func underline(_  
isActive: Bool = true, pattern:  
Text.LineStyle.Pattern = .solid, color:  
Color? = nil) -> some View
```

```
    /// Applies a strikethrough to the  
text in this view.  
    ///  
    /// - Parameters:  
    ///     - isActive: A Boolean value  
that indicates whether  
    ///         strikethrough is added. The  
default value is `true`.  
    ///     - pattern: The pattern of the  
line. The default value is `solid`.  
    ///     - color: The color of the  
strikethrough. If `color` is `nil`, the  
    ///         strikethrough uses the  
default foreground color.  
    ///
```

```
    /// - Returns: A view where text has
    // a line through its center.
    @available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
    nonisolated public func
strikethrough(_ isActive: Bool = true,
pattern: Text.LineStyle.Pattern = .solid,
color: Color? = nil) -> some View

}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension View {

    /// Adds a reason to apply a
    // redaction to this view hierarchy.
    ///
    /// Adding a redaction is an additive
    // process: any redaction
    /// provided will be added to the
    // reasons provided by the parent.
    nonisolated public func
redacted(reason: RedactionReasons) ->
some View

    /// Removes any reason to apply a
    // redaction to this view hierarchy.
    nonisolated public func unredacted()
-> some View

}
```

```
@available(iOS 15.0, macOS 10.15, watchOS 9.0, *)
@available(tvOS, unavailable)
extension View {

    /// Sets the size for controls within this view.
    ///
    /// Use `controlSize(_:)` to override the system default size for controls
    /// in this view. In this example, a view displays several typical controls
    /// at `.mini`, `.small` and `.regular` sizes.
    ///
    /// struct ControlSize: View {
    ///     var body: some View {
    ///         VStack {
    ///             MyControls(label:
    /// "Mini")
    ///             .controlSize(
    /// .mini)
    ///             MyControls(label:
    /// "Small")
    ///             .controlSize(
    /// .small)
    ///             MyControls(label:
    /// "Regular")
    ///             .controlSize(
    /// .regular)
    ///         }
    ///     }
}
```

```
    /// .frame(width: 450)
    /// .border(Color.gray)
    /// }
    /// }
    /// struct MyControls: View {
    ///     var label: String
    ///     @State private var value
= 3.0
    ///     @State private var
selected = 1
    ///         var body: some View {
    ///             HStack {
    ///                 Text(label + ":")
    ///
Picker("Selection", selection: $selected)
{
    ///             Text("option
1").tag(1)
    ///             Text("option
2").tag(2)
    ///             Text("option
3").tag(3)
    ///
    ///
    ///
    ///         Slider(value:
$value, in: 1...10)
    ///
    ///
    ///         Button("OK") { }
    ///
    ///
    ///     }
    ///
    ///
    ///     }
    ///
    ///
    /// ! [A screenshot showing several
controls of various
```

```
    /// sizes.] (SwiftUI-View-
controlSize.png)
    /**
     /**
      * - Parameter controlSize: One of
      * the control sizes specified in the
      * ``ControlSize`` enumeration.
      */
     @inlinable nonisolated public func
controlSize(_ controlSize: ControlSize)
-> some View

}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension View {

    /**
     * Sets an explicit active
     * appearance for materials in this view.
     */
    nonisolated public func
materialActiveAppearance(_ appearance:
MaterialActiveAppearance) -> some View

}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension View {

    /**
     * Sets the rendering mode for
     * symbol images within this view.
     */
    /**
     * - Parameter mode: The symbol
     * rendering mode to use.
     */
}
```

```
    /**
     * - Returns: A view that uses the
     * rendering mode you supply.
     */
    @inlinable nonisolated public func
    symbolRenderingMode(_ mode:
    SymbolRenderingMode?) -> some View

}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, *)
@available(watchOS, unavailable)
extension View {

    /**
     * Returns a new view that applies
     `shader` to `self` as a filter
     /**
     * effect on the color of each
     pixel.
    /**
     /**
     * For a shader function to act as a
     color filter it must have a
     /**
     * function signature matching:
    /**
     /**
     [[ stitchable ]] half4
    name(float2 position, half4 color,
    args...)
    /**
     /**
     * where `position` is the user-
     space coordinates of the pixel
    /**
     * applied to the shader and `color`
     its source color, as a
    /**
     * pre-multiplied color in the
     destination color space. `args...`
```

```
    /// should be compatible with the
uniform arguments bound to
    /// `shader`. The function should
return the modified color value.
    ///
    /// > Important: Views backed by
AppKit or UIKit views may not
    /// render into the filtered layer.
Instead, they log a warning
    /// and display a placeholder image
to highlight the error.
    ///
    /// - Parameters:
    ///   - shader: The shader to apply
to `self` as a color filter.
    ///   - isEnabled: Whether the effect
is enabled or not.
    ///
    /// - Returns: A new view that
renders `self` with the shader
    /// applied as a color filter.
nonisolated public func colorEffect(_
shader: Shader, isEnabled: Bool = true)
-> some View
```

```
    /// Returns a new view that applies
`shader` to `self` as a
    /// geometric distortion effect on
the location of each pixel.
    ///
    /// For a shader function to act as a
distortion effect it must
```

```
    /// have a function signature
matching:
    /**
     /**
      [[ stitchable ]] float2
name(float2 position, args...)
    /**
     /**
      where `position` is the user-
space coordinates of the
      /**
      destination pixel applied to the
      shader. `args...` should be
      /**
      compatible with the uniform
      arguments bound to `shader`. The
      /**
      function should return the user-
space coordinates of the
      /**
      corresponding source pixel.
    /**
    /**
      /**
      > Important: Views backed by
      AppKit or UIKit views may not
      /**
      render into the filtered layer.
      Instead, they log a warning
      /**
      and display a placeholder image
      to highlight the error.
    /**
    /**
      - Parameters:
    /**
      - shader: The shader to apply
      as a distortion effect.
    /**
      - maxSampleOffset: The maximum
      distance in each axis between
    /**
      the returned source pixel
      position and the destination pixel
    /**
      position, for all source
      pixels.
    /**
      - isEnabled: Whether the effect
```

is enabled or not.

```
///
/// - Returns: A new view that
renders `self` with the shader
/// applied as a distortion effect.
@nonisolated public func
distortionEffect(_ shader: Shader,
maxSampleOffset: CGSize, isEnabled: Bool
= true) -> some View
```

```
///
/// Returns a new view that applies
`shader` to `self` as a filter
/// on the raster layer created from
`self`.
///
/// For a shader function to act as a
layer effect it must
/// have a function signature
matching:
///
///     [[ stitchable ]] half4
name(float2 position,
      ///             SwiftUI::Layer layer,
args...)
///
/// where `position` is the user-
space coordinates of the
/// destination pixel applied to the
shader, and `layer` is a
/// subregion of the rasterized
contents of `self`. `args...`
/// should be compatible with the
```

```
uniform arguments bound to
    /// `shader`.
    ///
    /// The `SwiftUI::Layer` type is
defined in the
    /// `<SwiftUI/SwiftUI.h>` header
file. It exports a single
    /// `sample()` function that returns
a linearly-filtered pixel
    /// value from a position in the
source content, as a premultiplied
    /// RGBA pixel value:
    ///
    ///     namespace SwiftUI {
    ///         struct Layer {
    ///             half4 sample(float2
position) const;
    ///         };
    ///     };
    ///
    /// The function should return the
color mapping to the destination
    /// pixel, typically by sampling one
or more pixels from `layer` at
    /// location(s) derived from
`position` and then applying some kind
    /// of transformation to produce a
new color.
    ///
    /// > Important: Views backed by
AppKit or UIKit views may not
    /// render into the filtered layer.
Instead, they log a warning
```

```
    /// and display a placeholder image
    // to highlight the error.
    /**
     * - Parameters:
     *   - shader: The shader to apply
     *     as a layer effect.
     *   - maxSampleOffset: If the
     *     shader function samples from the
     *       layer at locations not equal
     *       to the destination position,
     *       this value must specify the
     *       maximum sampling distance in
     *       each axis, for all source
     *       pixels.
     *   - isEnabled: Whether the effect
     *     is enabled or not.
     */
    /**
     * - Returns: A new view that
     *     renders `self` with the shader
     *     applied as a distortion effect.
     */
    nonisolated public func layerEffect(_
shader: Shader, maxSampleOffset: CGSize,
isEnabled: Bool = true) -> some View
```

}

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension View {
```

/// Constrains this view's dimensions
 // to the specified aspect ratio.

//

```
    /// Use `aspectRatio(_:contentMode:)`  
    to constrain a view's dimensions to an  
    /// aspect ratio specified by a  
    ///  
<doc://com.apple.documentation/documentation/CoreFoundation/CGFloat>  
    /// using the specified content mode.  
    ///  
    /// If this view is resizable, the  
    resulting view will have `aspectRatio` as  
    /// its aspect ratio. In this  
example, the purple ellipse has a 3:4  
    /// width-to-height ratio, and scales  
to fit its frame:  
    ///  
    ///     Ellipse()  
    ///         .fill(Color.purple)  
    ///         .aspectRatio(0.75,  
contentMode: .fit)  
    ///         .frame(width: 200,  
height: 200)  
    ///         .border(Color.white:  
0.75))  
    ///  
    /// ! [A view showing a purple ellipse  
that has a 3:4 width-to-height ratio,  
    /// and scales to fit its frame.]  
(SwiftUI-View-aspectRatio-cgfloat.png)  
    ///  
    /// - Parameters:  
    ///     - aspectRatio: The ratio of  
width to height to use for the resulting  
    ///     view. Use `nil` to maintain
```

```
the current aspect ratio in the
    ///      resulting view.
    /// - contentMode: A flag that
indicates whether this view fits or fills
    ///      the parent context.
    ///
    /// - Returns: A view that constrains
this view's dimensions to the aspect
    ///      ratio of the given size using
`contentMode` as its scaling algorithm.
    @inlinable nonisolated public func
aspectRatio(_ aspectRatio: CGFloat? =
nil, contentMode: ContentMode) -> some
View
```

```
    /// Constrains this view's dimensions
to the aspect ratio of the given size.
    ///
    /// Use `aspectRatio(_:contentMode:)`
to constrain a view's dimensions to
    /// an aspect ratio specified by a
    ///
<doc://com.apple.documentation/documentation/CoreFoundation/CGSize>.
    ///
    /// If this view is resizable, the
resulting view uses `aspectRatio` as its
    /// own aspect ratio. In this
example, the purple ellipse has a 3:4
    /// width-to-height ratio, and scales
to fill its frame:
    ///
```

```
    ///      Ellipse()
    ///          .fill(Color.purple)
    ///          .aspectRatio(CGSize(width
: 3, height: 4), contentMode: .fill)
    ///          .frame(width: 200,
height: 200)
    ///          .border(Color.white:
0.75))
    ///
    /// ! [A view showing a purple ellipse
that has a 3:4 width-to-height ratio,
    /// and scales to fill its frame.]  

(SwiftUI-View-aspectRatio.png)
    ///
    /// - Parameters:
    ///   - aspectRatio: A size that
specifies the ratio of width to height to
    ///   use for the resulting view.
    ///   - contentMode: A flag
indicating whether this view should fit
or fill
    ///   the parent context.
    ///
    /// - Returns: A view that constrains
this view's dimensions to
    ///   `aspectRatio`, using
`contentMode` as its scaling algorithm.  

@inlinable nonisolated public func
aspectRatio(_ aspectRatio: CGSize,
contentMode: ContentMode) -> some View
```

/// Scales this view to fit its

```
parent.  
    ///  
    /// Use `scaledToFit()` to scale this  
view to fit its parent, while  
    /// maintaining the view's aspect  
ratio as the view scales.  
    ///  
    ///     Circle()  
    ///         .fill(Color.pink)  
    ///         .scaledToFit()  
    ///         .frame(width: 300,  
height: 150)  
    ///             .border(Color.white:  
0.75))  
    ///  
    ///     /// ! [A screenshot of pink circle  
scaled to fit its  
    /// frame.] (SwiftUI-View-  
scaledToFit-1.png)  
    ///  
    /// This method is equivalent to  
calling  
    ///  
``View/aspectRatio(_:contentMode:)`` with  
a `nil` aspectRatio and  
    /// a content mode of  
``ContentMode/fit``.  
    ///  
    /// - Returns: A view that scales  
this view to fit its parent, maintaining  
    /// this view's aspect ratio.  
@inlinable nonisolated public func  
scaledToFit() -> some View
```

```
    /// Scales this view to fill its
parent.
    /**
     /// Use `scaledToFill()` to scale
this view to fill its parent, while
     /// maintaining the view's aspect
ratio as the view scales:
    /**
     /**
         Circle()
     /**
         .fill(Color.pink)
     /**
         .scaledToFill()
     /**
         .frame(width: 300,
height: 150)
     /**
         .border(Color.white:
0.75)
    /**
        /**
            ! [A screenshot of pink circle
scaled to fill its
            /// frame.] (SwiftUI-View-
scaledToFill-1.png)
    /**
        /**
            This method is equivalent to
calling
    /**
        ``View/aspectRatio(_:contentMode:)`` with
a `nil` aspectRatio and
        /**
            a content mode of
        ``ContentMode/fill``.
    /**
        /**
            - Returns: A view that scales
this view to fill its parent, maintaining
```

```
    /// this view's aspect ratio.  
    @inlinable nonisolated public func  
scaledToFill() -> some View  
  
}  
  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension View {  
  
    /// Reads the specified preference  
value from the view, using it to  
    /// produce a second view that is  
applied as an overlay to the  
    /// original view.  
    ///  
    /// The values of the preference key  
from both views  
    /// are combined and made visible to  
the parent view.  
    ///  
    /// - Parameters:  
    ///     - key: The preference key type  
whose value is to be read.  
    ///     - alignment: An optional  
alignment to use when positioning the  
    ///         overlay view relative to the  
original view.  
    ///     - transform: A function that  
produces the overlay view from  
    ///         the preference value read  
from the original view.  
    ///
```

```
    /// - Returns: A view that layers a second view in front of the view.
```

```
    @inlinable nonisolated public func overlayPreferenceValue<K, V>(_ key: K.Type, alignment: Alignment = .center, @ViewBuilder _ transform: @escaping (K.Value) -> V) -> some View where K : PreferenceKey, V : View
```

```
    /// Reads the specified preference value from the view, using it to
```

```
        /// produce a second view that is applied as the background of the
```

```
        /// original view.
```

```
    ///
```

```
    /// The values of the preference key from both views
```

```
        /// are combined and made visible to the parent view.
```

```
    ///
```

```
    /// - Parameters:
```

```
        /// - key: The preference key type whose value is to be read.
```

```
        /// - alignment: An optional alignment to use when positioning the
```

```
            /// background view relative to the original view.
```

```
        /// - transform: A function that produces the background view from
```

```
            /// the preference value read from the original view.
```

```
    ///
```

```
    /// - Returns: A view that layers a
    /// second view behind the view.
    @inlinable nonisolated public func
backgroundPreferenceValue<K, V>(_ key:
K.Type, alignment: Alignment = .center,
@ViewBuilder _ transform: @escaping
(K.Value) -> V) -> some View where K : PreferenceKey, V : View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Reads the specified preference
    /// value from the view, using it to
    /// produce a second view that is
    /// applied as an overlay to the
    /// original view.
    ///
    /// - Parameters:
    ///   - key: The preference key type
    ///     whose value is to be read.
    ///   - transform: A function that
    ///     produces the overlay view from
    ///     the preference value read
    ///     from the original view.
    ///
    /// - Returns: A view that layers a
    /// second view in front of the view.
    @inlinable nonisolated public func
overlayPreferenceValue<Key, T>(_ key:
```

```
Key.Type = Key.self, @ViewBuilder _  
transform: @escaping (Key.Value) -> T) ->  
some View where Key : PreferenceKey, T :  
View
```

```
    /// Reads the specified preference  
    value from the view, using it to  
        /// produce a second view that is  
        applied as the background of the  
            /// original view.  
    ////  
    /// – Parameters:  
        /// – key: The preference key type  
        whose value is to be read.  
        /// – transform: A function that  
        produces the background view from  
            /// the preference value read  
        from the original view.  
    ////  
    /// – Returns: A view that layers a  
    second view behind the view.  
@inlinable nonisolated public func  
backgroundPreferenceValue<Key, T>(_ key:  
Key.Type = Key.self, @ViewBuilder _  
transform: @escaping (Key.Value) -> T) ->  
some View where Key : PreferenceKey, T :  
View  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
extension View {  
  
    /// Controls the display order of  
    /// overlapping views.  
    ///  
    /// Use `zIndex(_:)` when you want to  
    control the front-to-back ordering of  
    /// views.  
    ///  
    /// In this example there are two  
    overlapping rotated rectangles. The  
    /// frontmost is represented by the  
    larger index value.  
    ///  
    ///     VStack {  
    ///         Rectangle()  
    ///             .fill(Color.yellow)  
    ///             .frame(width: 100,  
height: 100, alignment: .center)  
    ///             .zIndex(1) // Top  
layer.  
    ///  
    ///         Rectangle()  
    ///             .fill(Color.red)  
    ///             .frame(width: 100,  
height: 100, alignment: .center)  
    ///             .rotationEffect(.degr  
ees(45))  
    ///                 // Here a zIndex of 0  
is the default making  
    ///                 // this the bottom  
layer.  
    /// }
```

```
    /**
     /// ! [A screenshot showing two
     overlapping rectangles. The frontmost
     view is
     /// represented by the larger zIndex
     value.] (SwiftUI-View-zIndex.png)
    /**
     /// - Parameter value: A relative
     front-to-back ordering for this view; the
     /// default is `0`.
     @inlinable nonisolated public func
     zIndex(_ value: Double) -> some View
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension View {

    /**
     /// Associates a value with a custom
     layout property.
    /**
     /// Use this method to set a value
     for a custom property that
     /// you define with
     ``LayoutValueKey``. For example, if you
     define
     /// a `Flexibility` key, you can set
     the key on a ``Text`` view
     /// using the key's type and a value:
    /**
     /**
     /**
         Text("Another View")
         .layoutValue(key:
```

```
Flexibility.self, value: 3)
    /**
     /// For convenience, you might define
     a method that does this in an
     /// extension to ``View``:
     /**
     ///     extension View {
     ///         func layoutFlexibility(_
value: CGFloat?) -> some View {
     ///             layoutValue(key:
Flexibility.self, value: value)
     ///         }
     ///     }
     /**
     /// This method makes the call site
     easier to read:
     /**
     ///     Text("Another View")
     ///         .layoutFlexibility(3)
     /**
     /// If you perform layout operations
     in a type that conforms to the
     /// ``Layout`` protocol, you can read
     the key's associated value for
     /// each subview of your custom
     layout type. Do this by indexing the
     /// subview's proxy with the key. For
     more information, see
     /// ``LayoutValueKey``.
     /**
     /// - Parameters:
     ///   - key: The type of the key that
     you want to set a value for.
```

```
    /// Create the key as a type that
conforms to the ``LayoutValueKey``
    /// protocol.
    /// - value: The value to assign to
the key for this view.
    /// The value must be of the type
that you establish for the key's
    /// associated value when you
implement the key's
    ///
``LayoutValueKey/defaultValue`` property.
    ///
    /// - Returns: A view that has the
specified value for the specified key.
    @inlinable nonisolated public func
layoutValue<K>(key: K.Type, value:
K.Value) -> some View where K :
LayoutValueKey

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Sets the alignment of a text view
that contains multiple lines of text.
    ///
    /// Use this modifier to set an
alignment for a multiline block of text.
    /// For example, the modifier centers
the contents of the following
    /// ``Text`` view:
```

```
///
///      Text("This is a block of text
that shows up in a text element as
multiple lines.\n") Here we have
chosen to center this text.")
///
///          .frame(width: 200)
///          .multilineTextAlignment(.center)
///
/// The text in the above example
spans more than one line because:
///
/// * The newline character
introduces a line break.
/// * The frame modifier limits the
space available to the text view, and
/// by default a text view wraps
lines that don't fit in the available
/// width. As a result, the text
before the explicit line break wraps to
/// three lines, and the text after
uses two lines.
///
/// The modifier applies the
alignment to all the lines of text in
/// the view, regardless of why
wrapping occurs:
///
/// ![A block of text that spans 5
lines. The lines of text are center-
aligned.](View-multilineTextAlignment-1-
iOS)
///
```

```
    /// The modifier has no effect on a
``Text`` view that contains only one
    /// line of text, because a text view
has a width that exactly matches the
    /// width of its widest line. If you
want to align an entire text view
    /// rather than its contents, set the
alignment of its container, like a
    /// ``VStack`` or a frame that you
create with the
    ///
``View/frame(minWidth:idealWidth:maxWidth
:minHeight:idealHeight:maxHeight:alignmen
t:)``
    /// modifier.
    ///
    /// > Note: You can use this modifier
to control the alignment of a ``Text``
    /// view that you create with the
``Text/init(_:style:)`` initializer
    /// to display localized dates and
times, including when the view uses
    /// only a single line, but only
when that view appears in a widget.
    ///
    /// The modifier also affects the
content alignment of other text container
    /// types, like ``TextEditor`` and
``TextField``. In those cases, the
    /// modifier sets the alignment even
when the view contains only a single
    /// line because view's width isn't
dictated by the width of the text it
```

```
    /// contains.  
    ///  
    /// The modifier operates by setting  
the  
    ///  
``EnvironmentValues/multilineTextAlignmen  
t`` value in the environment,  
    /// so it affects all the text  
containers in the modified view  
hierarchy.  
    /// For example, you can apply the  
modifier to a ``VStack`` to  
    /// configure all the text views  
inside the stack.  
    ///  
    /// - Parameter alignment: A value  
that you use to align multiple lines of  
    ///     text within a view.  
    ///  
    /// - Returns: A view that aligns the  
lines of multiline ``Text`` instances  
    ///     it contains.  
@inlinable nonisolated public func  
multilineTextAlignment(_ alignment:  
TextAlignment) -> some View
```

```
    /// Sets the truncation mode for  
lines of text that are too long to fit in  
    /// the available space.  
    ///  
    /// Use the `truncationMode(_:)`  
modifier to determine whether text in a
```

```
    /// long line is truncated at the
beginning, middle, or end. Truncation is
    /// indicated by adding an ellipsis
(...) to the line when removing text to
    /// indicate to readers that text is
missing.
    ///
    /// In the example below, the bounds
of text view constrains the amount of
    /// text that the view displays and
the `truncationMode(_:)` specifies from
    /// which direction and where to
display the truncation indicator:
    ///
    ///     Text("This is a block of text
that will show up in a text element as
multiple lines. The text will fill the
available space, and then, eventually, be
truncated.")
    ///
    ///         .frame(width: 150,
height: 150)
    ///
    ///         .truncationMode(.tail)
    ///
    /// ! [A screenshot showing the effect
of truncation mode on text in a
    /// view.] (SwiftUI-view-
truncationMode.png)
    ///
    /// - Parameter mode: The truncation
mode that specifies where to truncate
    /// the text within the text view,
if needed. You can truncate at the
    /// beginning, middle, or end of
```

the text view.

```
///  
/// – Returns: A view that truncates  
text at different points in a line  
/// depending on the mode you  
select.
```

```
@inlinable nonisolated public func  
truncationMode(_ mode:  
Text.TruncationMode) -> some View
```

```
/// Sets the amount of space between  
lines of text in this view.
```

```
///  
/// Use `lineSpacing(_:)` to set the  
amount of spacing from the bottom of  
/// one line to the top of the next  
for text elements in the view.
```

```
///  
/// In the ``Text`` view in the  
example below, 10 points separate the  
bottom
```

```
/// of one line to the top of the  
next as the text field wraps inside this  
/// view. Applying `lineSpacing(_:)`  
to a view hierarchy applies the line  
/// spacing to all text elements  
contained in the view.
```

```
///  
///     Text("This is a string in a  
TextField with 10 point spacing applied  
between the bottom of one line and the  
top of the next.")
```

```
    ///         .frame(width: 200,
height: 200, alignment: .leading)
    ///         .lineSpacing(10)
    ///
    /// ! [A screenshot showing the
effects of setting line spacing on the
text
    /// in a view.] (SwiftUI-view-
lineSpacing.png)
    ///
    /// - Parameter lineSpacing: The
amount of space between the bottom of one
    /// line and the top of the next
line in points.
@inlinable nonisolated public func
lineSpacing(_ lineSpacing: CGFloat) ->
some View

    /// Sets whether text in this view
can compress the space between characters
    /// when necessary to fit text in a
line.
    ///
    /// Use `allowsTightening(_:)` to
enable the compression of inter-character
    /// spacing of text in a view to try
to fit the text in the view's bounds.
    ///
    /// In the example below, two
identically configured text views show
the
    /// effects of `allowsTightening(_:)`
```

```
on the compression of the spacing
    /// between characters:
    ///
    ///     VStack {
    ///         Text("This is a wide text
element")
    ///             .font(.body)
    ///             .frame(width: 200,
height: 50, alignment: .leading)
    ///                 .lineLimit(1)
    ///                 .allowsTightening(true)
    ///
    ///         Text("This is a wide text
element")
    ///             .font(.body)
    ///             .frame(width: 200,
height: 50, alignment: .leading)
    ///                 .lineLimit(1)
    ///                 .allowsTightening(false)
    ///
    /// }
    ///
    /// ! [A screenshot showing the effect
of enabling text tightening in a
    /// view.] (SwiftUI-view-
allowsTightening.png)
    ///
    /// - Parameter flag: A Boolean value
that indicates whether the space
    /// between characters compresses
when necessary.
    ///
```

```
    /// - Returns: A view that can  
    compress the space between characters  
    when
```

```
    /// necessary to fit text in a  
    line.
```

```
    @inlinable nonisolated public func  
allowsTightening(_ flag: Bool) -> some  
View
```

```
    /// Sets the minimum amount that text  
    in this view scales down to fit in the
```

```
    /// available space.
```

```
    ///
```

```
    /// Use the `minimumScaleFactor(_:)`  
modifier if the text you place in a  
    /// view doesn't fit and it's okay if  
the text shrinks to accommodate. For
```

```
    /// example, a label with a minimum  
scale factor of `0.5` draws its text in  
    /// a font size as small as half of  
the actual font if needed.
```

```
    ///
```

```
    /// In the example below, the  
``HStack`` contains a ``Text`` label with  
a
```

```
    /// line limit of `1`, that is next  
to a ``TextField``. To allow the label  
    /// to fit into the available space,  
the `minimumScaleFactor(_:)` modifier  
    /// shrinks the text as needed to fit  
into the available space.
```

```
    ///
```

```
    ///      HStack {
    ///          Text("This is a long
label that will be scaled to fit:")
    ///              .lineLimit(1)
    ///              .minimumScaleFactor(0
.5)
    ///          TextField("My Long Text
Field", text: $myTextField)
    ///      }
    ///
    ///      !-[A screenshot showing the effect
of setting a minimumScaleFactor on
    /// text in a view.](SwiftUI-View-
minimumScaleFactor.png)
    ///
    /// - Parameter factor: A fraction
between 0 and 1 (inclusive) you use to
    /// specify the minimum amount of
text scaling that this view permits.
    ///
    /// - Returns: A view that limits the
amount of text downscaling.
@inlinable nonisolated public func
minimumScaleFactor(_ factor: CGFloat) ->
some View
```

```
    /// Sets a transform for the case of
the text contained in this view when
    /// displayed.
    ///
    /// The default value is `nil`,
displaying the text without any case
```

```
    /// changes.  
    ///  
    /// - Parameter textCase: One of the  
    ``Text/Case`` enumerations; the  
    /// default is `nil`.  
    /// - Returns: A view that transforms  
the case of the text.  
    @available(iOS 14.0, macOS 11.0, tvOS  
14.0, watchOS 7.0, *)  
    @inlinable nonisolated public func  
textCase(_ textCase: Text.Case?) -> some  
View  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Specifies the language for  
typesetting.  
    ///  
    /// In some cases `Text` may contain  
text of a particular language which  
    /// doesn't match the device UI  
language. In that case it's useful to  
    /// specify a language so line  
height, line breaking and spacing will  
    /// respect the script used for that  
language. For example:  
    ///  
    ///     Text(verbatim: "ແອປເປີລ")  
    ///             .typesettingLanguage(.ini
```

```
t(languageCode: .thai))  
    ///  
    /// Note: this language does not  
affect text localization.  
    ///  
    /// - Parameters:  
    ///   - language: The explicit  
language to use for typesetting.  
    ///   - isEnabled: A Boolean value  
that indicates whether text language is  
    ///     added  
    /// - Returns: A view with the  
typesetting language set to the value you  
    ///     supply.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
nonisolated public func  
typesettingLanguage(_ language:  
Locale.Language, isEnabled: Bool = true)  
-> some View
```

```
    /// Specifies the language for  
typesetting.  
    ///  
    /// In some cases `Text` may contain  
text of a particular language which  
    /// doesn't match the device UI  
language. In that case it's useful to  
    /// specify a language so line  
height, line breaking and spacing will  
    /// respect the script used for that  
language. For example:
```

```
    /**
     * Text(verbatim:
     * "แปลงเป็น").typesettingLanguage(
     *     .explicit(.init(languageCode: .thai)))
     *
     * Note: this language does not
     * affect text localized localization.
     *
     * - Parameters:
     *   - language: The language to use
     * for typesetting.
     *   - isEnabled: A Boolean value
     * that indicates whether text language is
     * added
     * - Returns: A view with the
     * typesetting language set to the value you
     * supply.
     @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
nonisolated public func
typesettingLanguage(_ language:
TypesettingLanguage, isEnabled: Bool =
true) -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Sets the maximum number of lines
    that text can occupy in this view.
```

```
///  
/// Use this modifier to cap the  
number of lines that an individual text  
/// element can display.  
///  
/// The line limit applies to all  
``Text`` instances within a hierarchy.  
For  
    /// example, an ``HStack`` with  
multiple pieces of text longer than three  
    /// lines caps each piece of text to  
three lines rather than capping the  
    /// total number of lines across the  
``HStack``.  
    ///  
    /// In the example below, the  
modifier limits the very long  
    /// line in the ``Text`` element to  
the 2 lines that fit within the view's  
    /// bounds:  
    ///  
    ///     Text("This is a long string  
that demonstrates the effect of SwiftUI's  
lineLimit(:_) operator.")  
        ///         .frame(width: 200,  
height: 200, alignment: .leading)  
        ///         .lineLimit(2)  
        ///  
        /// ! [A screenshot showing showing  
the effect of the line limit operator on  
        /// a very long string in a view.]  
(SwiftUI-view-lineLimit.png)  
    ///
```

```
    /// - Parameter number: The line
    limit. If `nil`, no line limit applies.
    ///
    /// - Returns: A view that limits the
    number of lines that ``Text``
    /// instances display.
    @available(iOS 13.0, macOS 10.15,
    tvOS 13.0, watchOS 6.0, *)
    @inlinable nonisolated public func
lineLimit(_ number: Int?) -> some View
```

```
    /// Sets to a partial range the
    number of lines that text can occupy in
    /// this view.
    ///
    /// Use this modifier to specify a
    partial range of lines that a ``Text``
    /// view or a vertical ``TextField``
    can occupy. When the text of such
    /// views occupies less space than
    the provided limit, that view expands to
    /// occupy the minimum number of
    lines.
    ///
    ///     Form {
    ///         TextField("Title", text:
    $model.title)
    ///         TextField("Notes", text:
    $model.notes, axis: .vertical)
    ///             .lineLimit(3...)
    ///     }
    ///
```

```
    /// - Parameter limit: The line
    limit.
    @available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
    nonisolated public func lineLimit(_
limit: PartialRangeFrom<Int>) -> some
View
```

```
    /// Sets to a partial range the
    number of lines that text can occupy
    /// in this view.
    /**
     * Use this modifier to specify a
     * partial range of lines that a
     * ``Text`` view or a vertical
     * ``TextField`` can occupy. When the text
     * of
     *   such views occupies more space
     * than the provided limit, a ``Text`` view
     *   truncates its content while a
     * ``TextField`` becomes scrollable.
    /**
     * Form {
     *   TextField("Title", text:
     * $model.title)
     *   TextField("Notes", text:
     * $model.notes, axis: .vertical)
     *   .lineLimit(...3)
     * }
    /**
     /// - Parameter limit: The line
    limit.
```

```
    @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
    nonisolated public func lineLimit(_  
limit: PartialRangeThrough<Int>) -> some  
View
```

```
        /// Sets to a closed range the number  
of lines that text can occupy in  
        /// this view.
```

```
        ///
```

```
        /// Use this modifier to specify a  
closed range of lines that a ``Text``  
        /// view or a vertical ``TextField``  
can occupy. When the text of such  
        /// views occupies more space than  
the provided limit, a ``Text`` view  
        /// truncates its content while a  
``TextField`` becomes scrollable.
```

```
        ///
```

```
        ///     Form {  
        ///         TextField("Title", text:  
$model.title)
```

```
        ///         TextField("Notes", text:  
$model.notes, axis: .vertical)  
        ///             .lineLimit(1...3)  
        /// }
```

```
        ///
```

```
        /// - Parameter limit: The line  
limit.
```

```
    @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
    nonisolated public func lineLimit(_
```

```
limit: ClosedRange<Int>) -> some View

    /// Sets a limit for the number of
lines text can occupy in this view.
    /**
     * Use this modifier to specify a
limit to the lines that a
     * ``Text`` or a vertical
``TextField`` may occupy. If passed a
     * value of true for the
`reservesSpace` parameter, and the text
of such
     * views occupies less space than
the provided limit, that view expands
     * to occupy the minimum number of
lines. When the text occupies
     * more space than the provided
limit, a ``Text`` view truncates its
     * content while a ``TextField``
becomes scrollable.
    /**
     * GroupBox {
     *     Text("Title")
     *         .font(.headline)
     *         .lineLimit(2,
reservesSpace: true)
     *         Text("Subtitle")
     *             .font(.subheadline)
     *             .lineLimit(4,
reservesSpace: true)
     *     }
    /**

```

```
    /// - Parameter limit: The line
    limit.
    /// - Parameter reservesSpace:
    Whether text reserves space so that
        /// it always occupies the height
    required to display the specified
        /// number of lines.
    @available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
    nonisolated public func lineLimit(_
limit: Int, reservesSpace: Bool) -> some
View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Sets the specified style to
    render backgrounds within the view.
    ///
    /// The following example uses this
    modifier to set the
    ///
    ``EnvironmentValues/backgroundStyle``
    environment value to a
        /// ``ShapeStyle/blue`` color that
    includes a subtle ``Color/gradient``.
    /// SwiftUI fills the ``Circle``
    shape that acts as a background element
        /// with this style:
    ///
```

```
    ///      Image(systemName: "swift")
    ///          .padding()
    ///          .background(in: Circle())
    ///          .backgroundStyle(.blue.gradient)
adient)
    ///
    /// ! [An image of the Swift logo
inside a circle that's blue with a slight
    /// linear gradient. The blue color
is slightly lighter at the top of the
    /// circle and slightly darker at the
bottom.] (View-backgroundStyle-1-iOS)
    ///
    /// To restore the default background
style, set the
    ///
``EnvironmentValues/backgroundStyle`` environment value to
    /// `nil` using the
``View/environment(_:_:)`` modifier:
    ///
    ///     .environment(\.backgroundStyle, nil)
    ///
    @available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
    @inlinable nonisolated public func
backgroundStyle<S>(_ style: S) -> some
View where S : ShapeStyle
}

@available(iOS 15.0, macOS 12.0, tvOS
```

```
15.0, watchOS 8.0, *)
extension View {

    /// Sets a view's foreground elements
    /// to use a given style.
    ///
    /// Use this method to style
    /// foreground content like text,
    shapes, and template images
    /// (including symbols):
    ///
    ///     HStack {
    ///         Image(systemName:
    "triangle.fill")
    ///         Text("Hello, world!")
    ///

    RoundedRectangle(cornerRadius: 5)
        ///
        .frame(width: 40,
height: 20)
    ///
    }

    /// The example above creates a row
of ``ShapeStyle/teal`` foreground
    /// elements:
    ///
    /// ! [A screenshot of a teal
triangle, string, and rounded
    /// rectangle.] (View-
foregroundStyle-1)
    ///
    /// You can use any style that
conforms to the ``ShapeStyle`` protocol,
```

```
    /// like the ``ShapeStyle/teal``  
color in the example above, or the  
    ///  
``ShapeStyle/linearGradient(colors:startP  
oint:endPoint:)`` gradient  
    /// shown below:  
    ///  
    ///     Text("Gradient Text")  
    ///         .font(.largeTitle)  
    ///         .foregroundStyle(  
    ///             .linearGradient(  
    ///                 colors: [.yellow,  
.blue],  
    ///                     startPoint: .top,  
    ///                     endPoint: .bottom  
    ///                 )  
    ///             )  
    ///  
    /// ! [A screenshot of the words  
Gradient Text, with letters that  
    /// appear yellow at the top, and  
transition to blue  
    /// toward the bottom.] (View-  
foregroundStyle-2)  
    ///  
    /// > Tip: If you want to fill a  
single ``Shape`` instance with a style,  
    /// use the ``Shape/fill(style:)``  
shape modifier instead because it's more  
    /// efficient.  
    ///  
    /// SwiftUI creates a context-  
dependent render for a given style.
```

```
    /// For example, a ``Color`` that you
    load from an asset catalog
    /// can have different light and dark
    appearances, while some styles
    /// also vary by platform.
    ///
    /// Hierarchical foreground styles
    like ``ShapeStyle/secondary``
    /// don't impose a style of their
    own, but instead modify other styles.
    /// In particular, they modify the
    primary
    /// level of the current foreground
    style to the degree given by
    /// the hierarchical style's name.
    /// To find the current foreground
    style to modify, SwiftUI looks for
    /// the innermost containing style
    that you apply with the
    /// `foregroundStyle(_:)` or the
    ``View/foregroundColor(_:)`` modifier.
    /// If you haven't specified a style,
    SwiftUI uses the default foreground
    /// style, as in the following
example:
    ///
    ///     VStack(alignment: .leading) {
    ///         Label("Primary",
    systemImage: "1.square.fill")
    ///         Label("Secondary",
    systemImage: "2.square.fill")
    ///             .foregroundStyle(.sec
ondary)
```

```
    /**
     */
    /**
     * ! [A screenshot of two labels with
     * the text primary and secondary.
     * The first appears in a brighter
     * shade than the
     * second, both in a grayscale
     * color.] (View-foregroundStyle-3)
     */
    /**
     * If you add a foreground style on
     * the enclosing
     * ``VStack``, the hierarchical
     * styling responds accordingly:
     */
    /**
     * VStack(alignment: .leading) {
     *     Label("Primary",
     * systemImage: "1.square.fill")
     *     Label("Secondary",
     * systemImage: "2.square.fill")
     *     .foregroundStyle(.sec
     * ondary)
     */
    }
    /**
     * .foregroundStyle(.blue)
     */
    /**
     * ! [A screenshot of two labels with
     * the text primary and secondary.
     * The first appears in a brighter
     * shade than the
     * second, both tinted blue.] (View-
     * foregroundStyle-4)
     */
    /**
     * When you apply a custom style to
     * a view, the view disables the vibrancy
```

```
    /// effect for foreground elements in
    // that view, or in any of its child
    // views, that it would otherwise
    gain from adding a background material
    // --- for example, using the
``View/background(_:ignoresSafeAreaEdges:
)``
    /// modifier. However, hierarchical
    styles applied to the default foreground
    /// don't disable vibrancy.
    /**
     * - Parameter style: The color or
     * pattern to use when filling in the
     * foreground elements. To
     * indicate a specific value, use ``Color``
     * or
     */
``ShapeStyle/image(_:sourceRect:scale:)``
, or one of the gradient
    /// types, like
    /**
     * - Returns: A view that uses the
     * given foreground style.
     */
``ShapeStyle/linearGradient(colors:startP
oint:endPoint:)``. To set a
    /// style that's relative to the
    containing view's style, use one of the
    /// semantic styles, like
``ShapeStyle/primary``.
    /**
     * @inlinable nonisolated public func
     * foregroundStyle<S>(_ style: S) -> some
     * View where S : ShapeStyle
```

```
    /// Sets the primary and secondary  
    levels of the foreground  
    /// style in the child view.  
    ///  
    /// SwiftUI uses these styles when  
    rendering child views  
    /// that don't have an explicit  
    rendering style, like images,  
    /// text, shapes, and so on.  
    ///  
    /// Symbol images within the view  
    hierarchy use the  
    /// ``SymbolRenderingMode/palette``  
    rendering mode when you apply this  
    /// modifier, if you don't explicitly  
    specify another mode.  
    ///  
    /// - Parameters:  
    ///     - primary: The primary color or  
    pattern to use when filling in  
    ///         the foreground elements. To  
    indicate a specific value, use ``Color``  
    ///         or  
    ``ShapeStyle/image(_:sourceRect:scale:)``  
    , or one of the gradient  
    ///         types, like  
    ///  
    ``ShapeStyle/linearGradient(colors:startP  
oint:endPoint:)``. To set a  
    ///         style that's relative to the  
containing view's style, use one of the
```

```
    /// semantic styles, like
``ShapeStyle/primary``.
    /// - secondary: The secondary
color or pattern to use when
    /// filling in the foreground
elements.
    ///
    /// - Returns: A view that uses the
given foreground styles.
@inlinable nonisolated public func
foregroundStyle<S1, S2>(_ primary: S1, -
secondary: S2) -> some View where S1 : -
ShapeStyle, S2 : ShapeStyle
```

```
    /// Sets the primary, secondary, and
tertiary levels of
    /// the foreground style.
    ///
    /// SwiftUI uses these styles when
rendering child views
    /// that don't have an explicit
rendering style, like images,
    /// text, shapes, and so on.
    ///
    /// Symbol images within the view
hierarchy use the
    /// ``SymbolRenderingMode/palette``
rendering mode when you apply this
    /// modifier, if you don't explicitly
specify another mode.
    ///
    /// - Parameters:
```

```
    /// - primary: The primary color or
    pattern to use when filling in
    /// the foreground elements. To
    indicate a specific value, use ``Color``
    /// or
``ShapeStyle/image(_:sourceRect:scale:)``
, or one of the gradient
    /// types, like
    ///
``ShapeStyle/linearGradient(colors:startP
oint:endPoint:)``. To set a
    /// style that's relative to the
containing view's style, use one of the
    /// semantic styles, like
``ShapeStyle/primary``.
    /// - secondary: The secondary
color or pattern to use when
    /// filling in the foreground
elements.
    /// - tertiary: The tertiary color
or pattern to use when
    /// filling in the foreground
elements.
    ///
    /// - Returns: A view that uses the
given foreground styles.
    @inlinable nonisolated public func
foregroundStyle<S1, S2, S3>(_ primary:
S1, _ secondary: S2, _ tertiary: S3) ->
some View where S1 : ShapeStyle, S2 :
ShapeStyle, S3 : ShapeStyle
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Applies a transformation to a  
    preference value.  
    @inlinable nonisolated public func  
    transformPreference<K>(_ key: K.Type =  
    K.self, _ callback: @escaping (inout  
    K.Value) -> Void) -> some View where K :  
    PreferenceKey  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Sets a value for the given  
    preference.  
    @inlinable nonisolated public func  
    preference<K>(key: K.Type = K.self,  
    value: K.Value) -> some View where K :  
    PreferenceKey  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Adds a luminance to alpha effect
```

to this view.

```
///
/// Use this modifier to create a
semitransparent mask, with the opacity of
/// each part of the modified view
controlled by the luminance of the
/// corresponding part of the
original view. Regions of lower luminance
/// become more transparent, while
higher luminance yields greater
/// opacity.
///
/// In particular, the modifier maps
the red, green, and blue components of
/// each input pixel's color to a
grayscale value, and that value becomes
/// the alpha component of a black
pixel in the output. This modifier
/// produces an effect that's
equivalent to using the `feColorMatrix`-
/// filter primitive with the
`luminanceToAlpha` type attribute, as
defined
/// by the [Scalable Vector Graphics
(SVG) 2](https://www.w3.org/TR/SVG2/)
/// specification.
///
/// The example below defines a
`Palette` view as a series of rectangles,
/// each composed as a ``Color`` with
a particular white value,
/// and then displays two versions of
the palette over a blue background:
```

```
///  
///     struct Palette: View {  
///         var body: some View {  
///             HStack(spacing: 0) {  
///                 ForEach(0..<10) {  
index in  
                    ///                         Color(white:  
Double(index) / Double(9))  
                    ///                         .frame(wi  
dth: 20, height: 40)  
                    ///                     }  
                    ///                     }  
                    ///                     }  
                    ///                     }  
                    ///                     }  
                    ///                     struct  
LuminanceToAlphaExample: View {  
    ///             var body: some View {  
    ///                 VStack(spacing: 20) {  
    ///                     Palette()  
    ///                     .padding()  
    ///                     .background(.blue)  
    ///                     }  
    ///                     }  
    ///                     }  
    ///                     /// The unmodified version of the  
palette contains rectangles that range  
    /// from solid black to solid white,
```

thus with increasing luminance. The  
    /// second version of the palette,  
which has the `luminanceToAlpha()``  
    /// modifier applied, allows the  
background to show through in an amount  
    /// that corresponds inversely to the  
luminance of the input.  
    ///  
    /// ! [A screenshot of a blue  
background with two wide rectangles on  
it,  
    /// arranged vertically, with one  
above the other. Each is composed of a  
    /// series of smaller rectangles  
arranged from left to right. The  
component  
    /// rectangles of the first large  
rectangle range from black to white as  
    /// you scan from left to right, with  
each successive component rectangle  
    /// slightly whiter than the  
previous. The component rectangles of the  
    /// second large rectangle range from  
fully transparent to fully opaque,  
    /// scanning in the same direction.]  
(View-luminanceToAlpha-1-iOS)  
    ///  
    /// - Returns: A view with the  
luminance to alpha effect applied.  
    **@inlinable nonisolated public func**  
**luminanceToAlpha() -> some View**  
}

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Adds an action to perform before  
    this view appears.  
    ///  
    /// The exact moment that SwiftUI  
    calls this method  
    /// depends on the specific view type  
    that you apply it to, but  
    /// the `action` closure completes  
    before the first  
    /// rendered frame appears.  
    ///  
    /// - Parameter action: The action to  
    perform. If `action` is `nil`, the  
    /// call has no effect.  
    ///  
    /// - Returns: A view that triggers  
    `action` before it appears.  
    @inlinable nonisolated public func  
    onAppear(perform action: (() -> Void)? =  
    nil) -> some View  
  
    /// Adds an action to perform after  
    this view disappears.  
    ///  
    /// The exact moment that SwiftUI  
    calls this method  
    /// depends on the specific view type
```

```
that you apply it to, but
    /// the `action` closure doesn't
execute until the view
    /// disappears from the interface.
    ///
    /// - Parameter action: The action to
perform. If `action` is `nil`, the
    /// call has no effect.
    ///
    /// - Returns: A view that triggers
`action` after it disappears.
    @inlinable nonisolated public func
onDisappear(perform action: (() -> Void)?
= nil) -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Transforms the environment value
of the specified key path with the
    /// given function.
    @inlinable nonisolated public func
transformEnvironment<V>(_ keyPath:
WritableKeyPath<EnvironmentValues, V>,
transform: @escaping (inout V) -> Void)
-> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
extension View {

    /// Sets the unique tag value of this
view.
    ///
    /// Use this modifier to
differentiate among certain selectable
views,
    /// like the possible values of a
``Picker`` or the tabs of a ``TabView``.
    /// Tag values can be of any type
that conforms to the
    ///
<doc://com.apple.documentation/documentation/Swift/Hashable> protocol.
    ///
    /// This modifier will write the tag
value for the type `V`, as well as
    /// `Optional<V>` if
`includeOptional` is enabled. Containers
checking for
    /// tags of either type will see the
value as set.
    ///
    /// In the example below, the
``ForEach`` loop in the ``Picker`` view
    /// builder iterates over the
`Flavor` enumeration. It extracts the
string
    /// value of each enumeration element
for use in constructing the row
    /// label, and uses the enumeration
```

```
value as input to the `tag(_:)`  
    /// modifier.  
    ///  
    ///     struct FlavorPicker: View {  
    ///         enum Flavor: String,  
CaseIterable, Identifiable {  
    ///             case chocolate,  
vanilla, strawberry  
    ///             var id: Self { self }  
    ///         }  
    ///  
    ///         @State private var  
selectedFlavor: Flavor? = nil  
    ///  
    ///         var body: some View {  
    ///             Picker("Flavor",  
selection: $selectedFlavor) {  
    ///                 ForEach(Flavor.allCases) { flavor in  
    ///                     Text(flavor.rawValue)  
    ///                         .tag(flavor)  
    ///                         }  
    ///                     }  
    ///                 }  
    ///             }  
    ///         }  
    ///     }  
    /// The selection type of the  
``Picker`` is an `Optional<Flavor>` and  
so it  
    /// will look for tags on its  
contents of `Optional<Flavor>` type. Since
```

the

```
    /// tag modifier defaults to having
`includeOptional` enabled, even though
    /// the tag for each option is a non-
optional `Flavor`, the tag modifier
    /// writes values for both the non-
optional, and optional versions of the
    /// value, allowing the contents to
be selectable by the ``Picker``.

    ///
    /// A ``ForEach`` automatically
applies a default tag to each enumerated
    /// view using the `id` parameter of
the corresponding element. If
    /// the element's `id` parameter and
the picker's `selection` input
    /// have exactly the same type, or
the same type but optional, you can omit
    /// the explicit tag modifier.

    ///
    /// To see examples that don't
require an explicit tag, see ``Picker``.

    ///
    /// - Parameter tag: A
<doc://com.apple.documentation/documentation/Swift/Hashable>
    ///     value to use as the view's tag.
    /// - Parameter includeOptional: If
the tag value for `Optional<V>` should
    ///     also be set.

    ///
    /// - Returns: A view with the
specified tag set.
```

```
    nonisolated public func tag<V>(_ tag:  
V, includeOptional: Bool = true) -> some  
View where V : Hashable
```

```
}
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension View {
```

```
    /// Sets the Dynamic Type size within  
the view to the given value.
```

```
    ///
```

```
    /// As an example, you can set a  
Dynamic Type size in `ContentView` to be  
    /// ``DynamicTypeSize/xLarge`` (this  
can be useful in previews to see your  
    /// content at a different size) like  
this:
```

```
    ///
```

```
    /// ContentView()
```

```
    ///     .dynamicTypeSize(.xLarge)
```

```
    ///
```

```
    /// If a Dynamic Type size range is  
applied after setting a value,
```

```
    /// the value is limited by that  
range:
```

```
    ///
```

```
    /// ContentView() // Dynamic Type  
size will be .large
```

```
    ///     .dynamicTypeSize(...Dynam  
icTypeSize.large)
```

```
    ///     .dynamicTypeSize(.xLarge)
```

```
    /**
     * When limiting the Dynamic Type
     * size, consider if adding a
     * large content view with
     * ``View/accessibilityShowsLargeContentView
     * er()``
     * would be appropriate.
     */
     * - Parameter size: The size to set
     * for this view.
     */
     * - Returns: A view that sets the
     * Dynamic Type size to the specified
     * `size`.
nonisolated public func
dynamicTypeSize(_ size: DynamicTypeSize)
-> some View
```

```
    /**
     * Limits the Dynamic Type size
     * within the view to the given range.
     */
     * As an example, you can constrain
     * the maximum Dynamic Type size in
     * `ContentView` to be no larger
     * than ``DynamicTypeSize/large``:
     */
     * ContentView()
     *     .dynamicTypeSize(...DynamicTypeSize.large)
     */
     * If the Dynamic Type size is
     * limited to multiple ranges, the result is
```

```
    /// their intersection:  
    ///  
    ///     ContentView() // Dynamic Type  
sizes are from .small to .large  
    ///         .dynamicTypeSize(.small..  
.)  
    ///         .dynamicTypeSize(...Dynam  
icTypeSize.large)  
    ///  
    /// A specific Dynamic Type size can  
still be set after a range is applied:  
    ///  
    ///     ContentView() // Dynamic Type  
size is .xLarge  
    ///         .dynamicTypeSize(.xLarge)  
    ///         .dynamicTypeSize(...Dynam  
icTypeSize.large)  
    ///  
    /// When limiting the Dynamic Type  
size, consider if adding a  
    /// large content view with  
``View/accessibilityShowsLargeContentView  
er()``  
    /// would be appropriate.  
    ///  
    /// - Parameter range: The range of  
sizes that are allowed in this view.  
    ///  
    /// - Returns: A view that constrains  
the Dynamic Type size of this view  
    /// within the specified `range`.  
nonisolated public func  
dynamicTypeSize<T>(_ range: T) -> some
```

```
View where T : RangeExpression, T.Bound  
== DynamicTypeSize  
  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension View {  
  
    /// Sets a particular container value  
    // of a view.  
    ///  
    /// Use this modifier to set one of  
    // the writable properties of the  
    /// ``ContainerValues`` structure,  
    // including custom values that you  
    /// create.  
    ///  
    /// Like preferences, container  
    // values are able to be read by views above  
    /// the view they're set on. Unlike  
    // preferences, however, container values  
    /// don't have merging behavior  
    // because they don't escape their closest  
    /// container. In the following  
    // example, the container value is set  
    /// on the contained view, but is  
    // dropped when it reaches the containing  
    /// ``VStack``.  
    ///  
    ///     VStack {  
    ///  
    Text("A").containerValue(\.myCustomValue,
```

```
1) // myCustomValue = 1
    /**
     Text("B").containerValue(\.myCustomValue,
2) // myCustomValue = 2
    /**
     // container values are
unaffected by views that aren't
containers:
    /**
     Text("C")
    /**
     .containerValue(\.myC
ustomValue, 3)
    /**
     .padding() //
myCustomValue = 3
    /**
     } // myCustomValue = it's
default value, values do not escape the
container
    /**
     /**
      Even if a stack has only one
child, container values still wouldn't
     /**
      be lifted to the `VStack`.
Container values don't escape a container
     /**
      even if the container has only
one child.
    /**
    /**
     - Parameters:
    /**
     - keyPath: A key path that
indicates the property of the
    /**
     ``ContainerValues`` structure
to update.
    /**
     - value: The new value to set
for the item specified by `keyPath`.
    /**
    /**
     - Returns: A view that has the
given value set in its containerValues.
```

```
    nonisolated public func
containerValue<V>(_ keyPath:
WritableKeyPath<ContainerValues, V>, _  
value: V) -> some View  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Sets a value for the specified  
    preference key, the value is a  
    /// function of a geometry value tied  
    to the current coordinate  
    /// space, allowing readers of the  
    value to convert the geometry to  
    /// their local coordinates.  
    ///  
    /// - Parameters:  
    ///     - key: the preference key type.  
    ///     - value: the geometry value in  
    the current coordinate space.  
    ///     - transform: the function to  
    produce the preference value.  
    ///  
    /// - Returns: a new version of the  
    view that writes the preference.  
    @inlinable nonisolated public func  
anchorPreference<A, K>(key _: K.Type =  
K.self, value: Anchor<A>.Source,  
transform: @escaping (Anchor<A>) ->  
K.Value) -> some View where K :
```

## PreferenceKey

}

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Positions this view within an  
    // invisible frame with the specified size.  
    ///  
    /// Use this method to specify a  
    // fixed size for a view's width, height, or  
    // both. If you only specify one of  
    // the dimensions, the resulting view  
    // assumes this view's sizing  
    behavior in the other dimension.  
    ///  
    /// For example, the following code  
    // lays out an ellipse in a fixed 200 by  
    // 100 frame. Because a shape always  
    occupies the space offered to it by  
    // the layout system, the first  
    // ellipse is 200x100 points. The second  
    // ellipse is laid out in a frame  
    with only a fixed height, so it occupies  
    // that height, and whatever width  
    the layout system offers to its parent.  
    ///  
    ///     VStack {  
    ///         Ellipse()  
    ///             .fill(Color.purple)  
    ///             .frame(width: 200,
```

```
height: 100)
    /**
     /// Ellipse()
     /// .fill(Color.blue)
     /// .frame(height: 100)
    /**
 */
    /**
     /// ! [A screenshot showing the effect
of frame size options: a purple
     /// ellipse shows the effect of a
fixed frame size, while a blue ellipse
     /// shows the effect of constraining
a view in one
     /// dimension.] (SwiftUI-View-
frame-1.png)
    /**
     /// `The alignment` parameter
specifies this view's alignment within
the
    /// frame.
    /**
    /**
     /// Text("Hello world!")
     /// .frame(width: 200,
height: 30, alignment: .topLeading)
     /// .border(Color.gray)
    /**
     /// In the example above, the text is
positioned at the top, leading corner
     /// of the frame. If the text is
taller than the frame, its bounds may
     /// extend beyond the bottom of the
frame's bounds.
    /**
    /**
     /// ! [A screenshot showing the effect
```

```
of frame size options on a text view
    /// showing a fixed frame size with a
specified
    /// alignment.] (SwiftUI-View-
frame-2.png)
    /**
     /**
     /**
     - Parameters:
     /**
     - width: A fixed width for the
resulting view. If `width` is `nil`,
     /**
     the resulting view assumes
this view's sizing behavior.
     /**
     - height: A fixed height for
the resulting view. If `height` is `nil`,
     /**
     the resulting view assumes
this view's sizing behavior.
     /**
     - alignment: The alignment of
this view inside the resulting frame.
     /**
     Note that most alignment
values have no apparent effect when the
     /**
     size of the frame happens to
match that of this view.
    /**
    /**
    /**
    - Returns: A view with fixed
dimensions of `width` and `height`, for
the
    /**
    parameters that are non-`nil`.
@inlinable nonisolated public func
frame(width: CGFloat? = nil, height:
CGFloat? = nil, alignment: Alignment
= .center) -> some View

    /**
    Positions this view within an
```

```
invisible frame.  
    ///  
    /// Use  
``SwiftUI/View/frame(width:height:alignme  
nt:)`` or  
    ///  
``SwiftUI/View/frame(minWidth:idealWidth:  
maxWidth:minHeight:idealHeight:maxHeight:  
alignment:)``  
    /// instead.  
    @available(*, deprecated, message:  
"Please pass one or more parameters."  
    @inlinable nonisolated public func  
frame() -> some View  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Positions this view within an  
    invisible frame having the specified size  
    /// constraints.  
    ///  
    /// Always specify at least one size  
    characteristic when calling this  
    /// method. Pass `nil` or leave out a  
    characteristic to indicate that the  
    /// frame should adopt this view's  
    sizing behavior, constrained by the other  
    /// non-`nil` arguments.  
    ///
```

/// The size proposed to this view is  
the size proposed to the frame,  
    /// limited by any constraints  
specified, and with any ideal dimensions  
    /// specified replacing any  
corresponding unspecified dimensions in  
the

    /// proposal.

    ///

    /// If no minimum or maximum  
constraint is specified in a given  
dimension,

    /// the frame adopts the sizing  
behavior of its child in that dimension.  
If

    /// both constraints are specified in  
a dimension, the frame unconditionally

    /// adopts the size proposed for it,  
clamped to the constraints. Otherwise,

    /// the size of the frame in either  
dimension is:

    ///

    /// - If a minimum constraint is  
specified and the size proposed for the  
    /// frame by the parent is less  
than the size of this view, the proposed  
    /// size, clamped to that minimum.

    /// - If a maximum constraint is  
specified and the size proposed for the  
    /// frame by the parent is greater  
than the size of this view, the  
    /// proposed size, clamped to that

maximum.

```
    /// - Otherwise, the size of this
    view.
    ///
    /// - Parameters:
    ///   - minWidth: The minimum width
    of the resulting frame.
    ///   - idealWidth: The ideal width
    of the resulting frame.
    ///   - maxWidth: The maximum width
    of the resulting frame.
    ///   - minHeight: The minimum height
    of the resulting frame.
    ///   - idealHeight: The ideal height
    of the resulting frame.
    ///   - maxHeight: The maximum height
    of the resulting frame.
    ///   - alignment: The alignment of
    this view inside the resulting frame.
    /// Note that most alignment
    values have no apparent effect when the
    /// size of the frame happens to
    match that of this view.
    ///
    /// - Returns: A view with flexible
    dimensions given by the call's non-'nil'
    /// parameters.
@inlinable nonisolated public func
frame(minWidth: CGFloat? = nil,
idealWidth: CGFloat? = nil, maxWidth:
CGFloat? = nil, minHeight: CGFloat? =
nil, idealHeight: CGFloat? = nil,
maxHeight: CGFloat? = nil, alignment:
Alignment = .center) -> some View
```

```
}
```

```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension View {  
  
    /// Returns a new view such that any  
    text views within it will use  
    /// `renderer` to draw themselves.  
    ///  
    /// - Parameter renderer: the  
    renderer value.  
    ///  
    /// - Returns: a new view that will  
    use `renderer` to draw its text  
    /// views.  
    nonisolated public func  
    textRenderer<T>(_ renderer: T) -> some  
    View where T : TextRenderer  
  
}
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {  
  
    /// Applies a text scale to text in  
    the view.  
    ///  
    /// - Parameters:  
    ///     - scale: The text scale to  
    apply.
```

```
    /// - isEnabled: If true the text
    scale is applied; otherwise text scale
    ///      is unchanged.
    /// - Returns: A view with the
    specified text scale applied.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    nonisolated public func textScale(_
scale: Text.Scale, isEnabled: Bool =
true) -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Applies the given animation to
    this view when the specified value
    /// changes.
    ///
    /// - Parameters:
    ///   - animation: The animation to
    apply. If `animation` is `nil`, the view
    ///      doesn't animate.
    ///   - value: A value to monitor for
    changes.
    ///
    /// - Returns: A view that applies
    `animation` to this view whenever `value`
    ///      changes.
    @inlinable nonisolated public func
animation<V>(_ animation: Animation?,
```

```
value: V) -> some View where V : Equatable

}

@available(iOS 15.0, macOS 12.0, tvOS 15.0, watchOS 8.0, *)
extension View where Self : Equatable {

    /// Applies the given animation to this view when this view changes.
    ///
    /// - Parameters:
    ///   - animation: The animation to apply. If `animation` is `nil`, the view
    ///     doesn't animate.
    ///
    /// - Returns: A view that applies `animation` to this view whenever it
    ///   changes.
    @available(iOS 15.0, macOS 12.0, tvOS 15.0, watchOS 8.0, *)
    @inlinable nonisolated public func animation(_ animation: Animation?) -> some View

}

@available(iOS 14.0, macOS 11.0, tvOS 14.0, watchOS 7.0, *)
extension View {

    /// Performs an action when a
```

specified value changes.

```
///  
/// Use this modifier to run a  
closure when a value like  
/// an ``Environment`` value or a  
``Binding`` changes.  
/// For example, you can clear a  
cache when you notice  
/// that the view's scene moves to  
the background:  
///  
///     struct ContentView: View {  
///  
@Environment(\.scenePhase) private var  
scenePhase  
    /// @StateObject private var  
cache = DataCache()  
    ///  
    ///         var body: some View {  
    ///             MyView()  
    ///             .onChange(of:  
scenePhase) { newScenePhase in  
        ///                 if  
newScenePhase == .background {  
            ///  
cache.empty()  
            ///         }  
            ///     }  
            /// }  
            ///  
    /// SwiftUI passes the new value into  
the closure. You can also capture the
```

```
    /// previous value to compare it to  
the new value. For example, in  
    /// the following code example,  
`PlayerView` passes both the old and new  
    /// values to the model.  
    ///  
    ///     struct PlayerView: View {  
    ///         var episode: Episode  
    ///         @State private var  
playState: PlayState = .paused  
    ///  
    ///         var body: some View {  
    ///             VStack {  
    ///  
Text(episode.title)  
    ///  
Text(episode.showTitle)  
    ///  
PlayButton(playState: $playState)  
    ///         }  
    ///         .onChange(of:  
playState) { [playState] newState in  
    ///  
model.playStateDidChange(from: playState,  
to: newState)  
    ///         }  
    ///     }  
    ///  
    ///     /// The system may call the action  
closure on the main actor, so avoid  
    /// long-running tasks in the  
closure. If you need to perform such
```

```
tasks,
    /// detach an asynchronous background
task.
    /**
     /// Important: This modifier is
     deprecated and has been replaced with new
     /// versions that include either zero
     or two parameters within the closure,
     /// unlike this version that includes
     one parameter. This deprecated version
     /// and the new versions behave
     differently with respect to how they
     execute
     /// the action closure, specifically
     when the closure captures other values.
     /// Using the deprecated API, the
     closure is run with captured values that
     /// represent the "old" state. With
     the replacement API, the closure is run
     /// with captured values that
     represent the "new" state, which makes it
     /// easier to correctly perform
     updates that rely on supplementary values
     /// (that may or may not have
     changed) in addition to the changed value
     that
     /// triggered the action.
    /**
     /// - Important: This modifier is
     deprecated and has been replaced with new
     /// versions that include either
     zero or two parameters within the
     /// closure, unlike this version
```

that includes one parameter. This

    /// deprecated version and the new  
    versions behave differently with

    /// respect to how they execute the  
    action closure, specifically when the

    /// closure captures other values.

Using the deprecated API, the closure

    /// is run with captured values  
    that represent the "old" state. With the

    /// replacement API, the closure is  
    run with captured values that

    /// represent the "new" state,  
    which makes it easier to correctly  
    perform

    /// updates that rely on  
    supplementary values (that may or may not  
    have

    /// changed) in addition to the  
    changed value that triggered the action.

    ///

    /// - Parameters:

        /// - value: The value to check  
        when determining whether to run the

        /// closure. The value must  
        conform to the

        ///

<doc://com.apple.documentation/documentation/Swift/Equatable>

    /// protocol.

    /// - action: A closure to run when  
    the value changes. The closure

        /// takes a `newValue` parameter  
        that indicates the updated

```
    ///      value.  
    ///  
    /// - Returns: A view that runs an  
    /// action when the specified value changes.  
    @available(iOS, deprecated: 17.0,  
message: "Use `onChange` with a two or  
zero parameter action closure instead.")  
    @available(macOS, deprecated: 14.0,  
message: "Use `onChange` with a two or  
zero parameter action closure instead.")  
    @available(tvOS, deprecated: 17.0,  
message: "Use `onChange` with a two or  
zero parameter action closure instead.")  
    @available(watchOS, deprecated: 10.0,  
message: "Use `onChange` with a two or  
zero parameter action closure instead.")  
    @available(visionOS, deprecated: 1.0,  
message: "Use `onChange` with a two or  
zero parameter action closure instead.")  
    @inlinable nonisolated public func  
onChange<V>(of value: V, perform action:  
@escaping (_ newValue: V) -> Void) ->  
some View where V : Equatable  
  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension View {  
  
    /// Adds a modifier for this view  
    /// that fires an action when a specific  
    /// value changes.
```

```
///  
/// You can use `onChange` to trigger  
a side effect as the result of a  
/// value changing, such as an  
`Environment` key or a `Binding`.  
///  
/// The system may call the action  
closure on the main actor, so avoid  
/// long-running tasks in the  
closure. If you need to perform such  
tasks,  
/// detach an asynchronous background  
task.  
///  
/// When the value changes, the new  
version of the closure will be called,  
/// so any captured values will have  
their values from the time that the  
/// observed value has its new value.  
The old and new observed values are  
/// passed into the closure. In the  
following code example, `PlayerView`  
/// passes both the old and new  
values to the model.  
///  
///     struct PlayerView: View {  
///         var episode: Episode  
///         @State private var  
playState: PlayState = .paused  
///  
///             var body: some View {  
///                 VStack {  
///
```



```
    nonisolated public func  
onChange<V>(of value: V, initial: Bool =  
false, _ action: @escaping (_ oldValue:  
V, _ newValue: V) -> Void) -> some View  
where V : Equatable
```

```
    /// Adds a modifier for this view  
that fires an action when a specific  
    /// value changes.  
    ///  
    /// You can use `onChange` to trigger  
a side effect as the result of a  
    /// value changing, such as an  
`Environment` key or a `Binding`.  
    ///  
    /// The system may call the action  
closure on the main actor, so avoid  
    /// long-running tasks in the  
closure. If you need to perform such  
tasks,  
    /// detach an asynchronous background  
task.  
    ///  
    /// When the value changes, the new  
version of the closure will be called,  
    /// so any captured values will have  
their values from the time that the  
    /// observed value has its new value.  
In the following code example,  
    /// `PlayerView` calls into its model  
when `playState` changes model.  
    ///
```

```
///      struct PlayerView: View {
///          var episode: Episode
///          @State private var
playState: PlayState = .paused
///
///          var body: some View {
///              VStack {
///
Text(episode.title)
///
Text(episode.showTitle)
///
PlayButton(playState: $playState)
///
///
///
model.playStateDidChange(state:
playState) {
    ///
model.playStateDidChange(state:
playState)
    ///
    ///
    ///
    ///
    ///
    /// - Parameters:
    ///     - value: The value to check
against when determining whether
    ///
    to run the closure.
    ///
    - initial: Whether the action
should be run when this view initially
    ///
    appears.
    ///
    - action: A closure to run when
the value changes.
///
```

```
    /// - Returns: A view that fires an
    // action when the specified value changes.
    nonisolated public func
onChange<V>(of value: V, initial: Bool =
false, _ action: @escaping () -> Void) ->
some View where V : Equatable

}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension View {

    /// Sets the container shape to use
    // for any container relative shape
    /// within this view.
    ///
    /// The example below defines a view
    // that shows its content with a rounded
    /// rectangle background and the same
    // container shape. Any
    /// ``ContainerRelativeShape`` within
    the `content` matches the rounded
    /// rectangle shape from this
    // container inset as appropriate.
    ///
    /// struct
PlatterContainer<Content: View> : View {
    ///             @ViewBuilder var content:
Content
    ///             var body: some View {
    ///             content
    ///             .padding()
```

```
    /**
     * .containerShape(s
     * shape)
     /**
     * .background(shape
     * .fill(.background))
     /**
     */
     var shape:
RoundedRectangle
{ RoundedRectangle(cornerRadius: 20) }
    /**
     */
    /**
     * @inlinable nonisolated public func
     * containerShape<T>(_ shape: T) -> some
     * View where T : InsettableShape
}

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

    /**
     * Animates effects that you apply
     * to a view over a sequence of phases
     * that change based on a trigger.
     */
    /**
     * When the modified view first
     * appears, this modifier renders its
     * `content` closure using the first
     * phase as input to the closure,
     * along with a proxy for the
     * modified view. Apply effects to the
     * proxy --- and thus to the
     * modified view --- in a way that's
     * appropriate
}
```

```
    /// for the first phase value.  
    ///  
    /// Later, when the value of the  
`trigger` input changes, the modifier  
    /// provides its `content` closure  
with the value of the second phase.  
    /// Update the effects that you apply  
to the proxy view accordingly,  
    /// and the modifier animates the  
change for you. The next time the  
    /// `trigger` input changes, this  
procedure repeats using successive phases  
    /// until reaching the last phase, at  
which point the modifier loops back  
    /// to the first phase.  
    ///  
    /// - Parameters:  
    /// - phases: The sequence of  
phases to cycle through. Ensure that the  
    /// sequence isn't empty. If it  
is, SwiftUI logs a runtime warning and  
    /// also returns a visual warning  
as the output view.  
    /// - trigger: A value whose  
changes cause the animator to use the  
    /// next phase.  
    /// - content: A view builder  
closure that takes two parameters: a  
proxy  
    ///     value representing the  
modified view and the current phase.  
    /// You can apply effects to the  
proxy based on the current phase.
```

```
    /// - animation: A closure that
    takes the current phase as input. Return
    /// the animation to use when
    transitioning to the next phase. If you
    /// return `nil`, the transition
    doesn't animate. If you don't set this
    /// parameter, SwiftUI uses a
    default animation.
```

```
nonisolated public func
phaseAnimator<Phase>(_ phases: some
Sequence, trigger: some Equatable,
@ViewBuilder content: @escaping
(PlaceholderContentView<Self>, Phase) ->
some View, animation: @escaping (Phase)
-> Animation? = { _ in .default }) ->
some View where Phase : Equatable
```

```
    /// Animates effects that you apply
    to a view over a sequence of phases
    /// that change continuously.
    ///
    /// When the modified view first
    appears, this modifier renders its
    /// `content` closure using the first
    phase as input to the closure,
    /// along with a proxy for the
    modified view. Apply effects to the
    /// proxy --- and thus to the
    modified view --- in a way that's
    appropriate
    /// for the first phase value.
    ///
```

```
    /// Right away, the modifier provides
    its `content` closure with the value
    /// of the second phase. Update the
    effects that you apply to the proxy
    /// view accordingly, and the
    modifier animates the change for you.
    /// As soon as the animation
    completes, the procedure repeats using
    /// successive phases until reaching
    the last phase, at which point the
    /// modifier loops back to the first
    phase.

    ///
    /// - Parameters:
    ///   - phases: The sequence of
    phases to cycle through. Ensure that the
    ///   sequence isn't empty. If it
    is, SwiftUI logs a runtime warning and
    ///   also returns a visual warning
    as the output view.
    ///   - content: A view builder
    closure that takes two parameters: a
    proxy
    ///     value representing the
    modified view and the current phase.
    ///     You can apply effects to the
    proxy based on the current phase.
    ///   - animation: A closure that
    takes the current phase as input. Return
    ///     the animation to use when
    transitioning to the next phase. If you
    ///     return `nil`, the transition
    doesn't animate. If you don't set this
```

```
    /// parameter, SwiftUI uses a
    default animation.
    nonisolated public func
phaseAnimator<Phase>(_ phases: some
Sequence, @ViewBuilder content: @escaping
(PlaceholderContentView<Self>, Phase) ->
some View, animation: @escaping (Phase)
-> Animation? = { _ in .default }) ->
some View where Phase : Equatable

}

@available(iOS 13.0, macOS 10.15, tvOS
16.0, watchOS 6.0, *)
extension View {

    /// Adds an action to perform when
this view recognizes a tap gesture.
    ///
    /// Use this method to perform the
specified `action` when the user clicks
    /// or taps on the view or container
`count` times.
    ///
    /// > Note: If you create a control
that's functionally equivalent
    /// to a ``Button``, use
``ButtonStyle`` to create a customized
button
    /// instead.
    ///
    /// In the example below, the color
of the heart images changes to a random
```

```
    /// color from the `colors` array
    whenever the user clicks or taps on the
    /// view twice:
    ///
    ///     struct TapGestureExample:
View {
    ///         let colors: [Color] =
    [.gray, .red, .orange, .yellow,
    ///         .g
reen, .blue, .purple, .pink]
    ///         @State private var
fgColor: Color = .gray
    ///
    ///         var body: some View {
    ///             Image(systemName:
"heart.fill")
    ///                 .resizable()
    ///                 .frame(width:
200, height: 200)
    ///                 .foregroundColor(
fgColor)
    ///                 .onTapGesture(cou
nt: 2) {
    ///                     fgColor =
colors.randomElement()!
    ///                 }
    ///             }
    ///         }
    ///
    ///         ! [A screenshot of a view of a
heart.] (SwiftUI-View-TapGesture.png)
    ///
    /// - Parameters:
```

```
    /// - count: The number of taps or
clicks required to trigger the action
    /// closure provided in
`action`. Defaults to `1`.
    /// - action: The action to
perform.

    nonisolated public func
onTapGesture(count: Int = 1, perform
action: @escaping () -> Void) -> some
View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Sets the priority by which a
parent layout should apportion space to
    /// this child.
    ///
    /// Views typically have a default
priority of `0` which causes space to be
    /// apportioned evenly to all sibling
views. Raising a view's layout
    /// priority encourages the higher
priority view to shrink later when the
    /// group is shrunk and stretch
sooner when the group is stretched.
    ///
    /// HStack {
    ///     Text("This is a
moderately long string.")
```

```
///         .font(.largeTitle)
///         .border(Color.gray)
///
///         Spacer()
///
///         Text("This is a higher
priority string.")
///
///         .font(.largeTitle)
///         .layoutPriority(1)
///         .border(Color.gray)
///
///     }
///
/// In the example above, the first
``Text`` element has the default
/// priority `0` which causes its
view to shrink dramatically due to the
/// higher priority of the second
``Text`` element, even though all of
their
/// other attributes (font, font size
and character count) are the same.
///
/// ![A screenshot showing twoText
views different layout
/// priorities.](SwiftUI-View-
layoutPriority.png)
///
/// A parent layout offers the child
views with the highest layout priority
/// all the space offered to the
parent minus the minimum space required
for
/// all its lower-priority children.
```

```
    /**
     * - Parameter value: The priority
     * by which a parent layout apportions
     * space to the child.
     */
    @inlinable nonisolated public func
    layoutPriority(_ value: Double) -> some
    View

}

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

    /**
     * Places an observable object in
     * the view's environment.
     */
    /**
     * Use this modifier to place an
     * object that you declare with the
     */
    <doc://com.apple.documentation/documentation/Observation/Observable()>
    /**
     * macro into a view's environment.
     * For example, you can add an instance
     * of a custom observable `Profile`
     * class to the environment of a
     * `ContentView`:
     */
    /**
     *     @Observable class Profile
    { ... }
    /**
     *     struct RootView: View {
     *         @State private var
```

```
currentProfile: Profile?  
    ///  
    ///         var body: some View {  
    ///             ContentView()  
    ///                 .environment(curr  
entProfile)  
    ///                     }  
    ///     }  
    ///  
    /// You then read the object inside  
`ContentView` or one of its descendants  
    /// using the ``Environment``  
property wrapper:  
    ///  
    ///     struct ContentView: View {  
    ///  
@Environment(Profile.self) private var  
currentProfile: Profile  
    ///  
    ///         var body: some View { ...  
}  
    ///     }  
    ///  
    /// This modifier affects the given  
view, as well as that view's descendant  
    /// views. It has no effect outside  
the view hierarchy on which you call it.  
    /// The environment of a given view  
hierarchy holds only one observable  
    /// object of a given type.  
    ///  
    /// - Note: This modifier takes an  
object that conforms to the
```

```
    /**
<doc://com.apple.documentation/documentation/Observation/Observable>
    /// protocol. To add environment
objects that conform to the
    /**
<doc://com.apple.documentation/documentation/Combine/ObservableObject>
    /// protocol, use
``View/environmentObject(_:)`` instead.
    /**
    /// - Parameter object: The object to
set for this object's type in the
    /// environment, or `nil` to clear
an object of this type from the
    /// environment.
    /**
    /// - Returns: A view that has the
specified object in its environment.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    nonisolated public func
environment<T>(_ object: T?) -> some View
where T : AnyObject, T : Observable

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Adds an action to perform when
the specified preference key's value
```

```
    /// changes.  
    ///  
    /// - Parameters:  
    ///   - key: The key to monitor for  
    value changes.  
    ///   - action: The action to perform  
    when the value for `key` changes. The  
    ///     `action` closure passes the  
    new value as its parameter.  
    ///  
    /// - Returns: A view that triggers  
    `action` when the value for `key`  
    /// changes.  
    @preconcurrency @inlinable  
nonisolated public func  
onPreferenceChange<K>(_ key: K.Type =  
K.self, perform action: @escaping  
@Sendable (K.Value) -> Void) -> some View  
where K : PreferenceKey, K.Value :  
Equatable  
  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension View {  
  
    /// Sets the header prominence for  
    this view.  
    ///  
    /// In the following example, the  
    section header appears with increased  
    /// prominence:
```

```
    /**
     *      List {
     *          Section(header:
Text("Header")) {
     *              Text("Row")
     *          }
     *          .headerProminence(.increa
sed)
     *      }
     *      .listStyle(.insetGrouped)
     *
     * - Parameter prominence: The
prominence to apply.
nonisolated public func
headerProminence(_ prominence:
Prominence) -> some View
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Applies a modifier to a view and
returns a new view.
    /**
     * Use this modifier to combine a
``View`` and a ``ViewModifier``, to
     * create a new view. For example,
if you create a view modifier for
     * a new kind of caption with blue
text surrounded by a rounded rectangle:
    */
```

```
    /// struct BorderedCaption:  
ViewModifier {  
    /// func body(content:  
Content) -> some View {  
        /// content  
        /// .font(.caption2)  
        /// .padding(10)  
        /// .overlay(  
        ///  
        /// RoundedRectangle(cornerRadius: 15)  
        /// .stroke(l  
ineWidth: 1)  
        /// )  
        /// .foregroundColor(  
Color.blue)  
        /// }  
        /// }  
        ///  
        /// You can use ``modifier(_:)`` to  
extend ``View`` to create new modifier  
        /// for applying the  
`BorderedCaption` defined above:  
        ///  
        /// extension View {  
        ///     func borderedCaption() ->  
some View {  
        ///  
modifier(BorderedCaption())  
        /// }  
        /// }  
        ///  
        /// Then you can apply the bordered  
caption to any view:
```

```
///  
///     Image(systemName: "bus")  
///         .resizable()  
///         .frame(width:50,  
height:50)  
///     Text("Downtown Bus")  
///         .borderedCaption()  
///  
///     /// ! [A screenshot showing the image  
of a bus with a caption reading  
    /// Downtown Bus. A view extension,  
using custom a modifier, renders the  
    /// caption in blue text surrounded  
by a rounded  
    /// rectangle.] (SwiftUI-View-  
ViewModifier.png)  
///  
/// - Parameter modifier: The  
modifier to apply to this view.  
@inlinable nonisolated public func  
modifier<T>(_ modifier: T) ->  
ModifiedContent<Self, T>  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, *)  
@available(watchOS, unavailable)  
extension View {  
  
    /// Returns a new view configured  
with the specified allowed  
    /// dynamic range.  
    ///
```

```
    /// The following example enables HDR
    // rendering within a view
    /// hierarchy:
    ///
    ///
MyView().allowedDynamicRange(.high)
    ///
    /// - Parameter range: the requested
    dynamic range, or nil to
    ///     restore the default allowed
    range.
    ///
    /// - Returns: a new view.
    nonisolated public func
allowedDynamicRange(_ range:
Image.DynamicRange?) -> some View

}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension View {

    /// Makes symbols within the view
    show a particular variant.
    ///
    /// When you want all the
    ///
<doc://com.apple.documentation/design/hum
an-interface-guidelines/sf-symbols>
    /// in a part of your app's user
    interface to use the same variant, use
    the
```

```
    /// `symbolVariant(_:)` modifier with
a ``SymbolVariants`` value, like
    /// ``SymbolVariants/fill-
swift.type.property``:
    ///
    ///     VStack(spacing: 20) {
    ///         HStack(spacing: 20) {
    ///             Image(systemName:
"person")
    ///             Image(systemName:
"folder")
    ///             Image(systemName:
"gearshape")
    ///             Image(systemName:
"list.bullet")
    ///         }
    ///     }
    ///     HStack(spacing: 20) {
    ///         Image(systemName:
"person")
    ///         Image(systemName:
"folder")
    ///         Image(systemName:
"gearshape")
    ///         Image(systemName:
"list.bullet")
    ///     }
    ///     .symbolVariant(.fill) // Shows filled variants, when available.
    ///
    ///     /// A symbol that doesn't have the specified variant remains unaffected.
```

```
    /// In the example above, the
`list.bullet` symbol doesn't have a
filled
    /// variant, so the
`symbolVariant(_:)` modifier has no
effect.
    ///
    /// ! [A screenshot showing two rows
of four symbols. Both rows contain a
    /// person, a folder, a gear, and a
bullet list. The symbols in the first
    /// row are outlined. The symbols in
the second row are filled, except the
    /// list, which is the same in both
rows.] (View-symbolVariant-1)
    ///
    /// If you apply the modifier more
than once, its effects accumulate.
    /// Alternatively, you can apply
multiple variants in one call:
    ///
    ///     Label("Airplane",
systemImage: "airplane.circle.fill")
    ///
    ///     Label("Airplane",
systemImage: "airplane")
    ///             .symbolVariant(.circle)
    ///             .symbolVariant(.fill)
    ///
    ///     Label("Airplane",
systemImage: "airplane")
    ///             .symbolVariant(.circle.fi
ll)
```

```
///  
/// All of the labels in the code  
above produce the same output:  
///  
/// ! [A screenshot of a label that  
shows an airplane in a filled circle  
/// beside the word Airplane.] (View-  
symbolVariant-2)  
///  
/// You can apply all these variants  
in any order, but  
/// if you apply more than one shape  
variant, the one closest to the  
/// symbol takes precedence. For  
example, the following image uses the  
/// ``SymbolVariants/square-  
swift.type.property`` shape:  
///  
///     Image(systemName:  
"arrow.left")  
///             .symbolVariant(.square) /  
/ This shape takes precedence.  
///             .symbolVariant(.circle)  
///             .symbolVariant(.fill)  
///  
/// ! [A screenshot of a left arrow  
symbol in a filled  
/// square.] (View-symbolVariant-3)  
///  
/// To cause a symbol to ignore the  
variants currently in the environment,  
/// directly set the  
``EnvironmentValues/symbolVariants``
```

```
environment value
    /// to ``SymbolVariants/none`` using
the ``View/environment(_:_:)`` modifier.
    ///
    /// - Parameter variant: The variant
to use for symbols. Use the values in
    /// ``SymbolVariants``.
    /// - Returns: A view that applies
the specified symbol variant or variants
    /// to itself and its child views.
nonisolated public func
symbolVariant(_ variant: SymbolVariants)
-> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Layers the given view behind this
view.
    ///
    /// Use `background(_:alignment:)`
when you need to place one view behind
    /// another, with the background view
optionally aligned with a specified
    /// edge of the frontmost view.
    ///
    /// The example below creates two
views: the `Frontmost` view, and the
    /// `DiamondBackground` view. The
`Frontmost` view uses the
```

```
    /// `DiamondBackground` view for the
background of the image element inside
    /// the `Frontmost` view's
``VStack``.
    ///
    /// struct DiamondBackground:
View {
    ///
        var body: some View {
    ///
        VStack {
    ///
            Rectangle()
    ///
                .fill(.gray)
    ///
                .frame(width:
250, height: 250, alignment: .center)
    ///
                    .rotationEffe
ct(.degrees(45.0))
    ///
        }
    ///
        }
    ///
}
    /// struct Frontmost: View {
    ///     var body: some View {
    ///         VStack {
    ///             Image(systemName:
"folder")
    ///
                    .font(.system
(size: 128, weight: .ultraLight))
    ///
                    .background(D
iamondBackground()))
    ///
        }
    ///
        }
    ///
}
    /// ! [A view showing a large folder
```

image with a gray diamond placed behind  
    /// it as its background view.] (View-  
background-1)

    ///

    /// - Parameters:

    /// - background: The view to draw  
behind this view.

    /// - alignment: The alignment with  
a default value of

    ///     ``Alignment/center`` that you  
use to position the background view.

    @available(iOS, introduced: 13.0,  
deprecated: 100000.0, message: "Use  
`background(alignment:content:)`  
instead.")

    @available(macOS, introduced: 10.15,  
deprecated: 100000.0, message: "Use  
`background(alignment:content:)`  
instead.")

    @available(tvOS, introduced: 13.0,  
deprecated: 100000.0, message: "Use  
`background(alignment:content:)`  
instead.")

    @available(watchOS, introduced: 6.0,  
deprecated: 100000.0, message: "Use  
`background(alignment:content:)`  
instead.")

    @available(visionOS, introduced: 1.0,  
deprecated: 100000.0, message: "Use  
`background(alignment:content:)`  
instead.")

    @inlinable nonisolated public func  
background<Background>(\_ background:

```
Background, alignment: Alignment
= .center) -> some View where
Background : View

}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension View {

    /// Layers the views that you specify
    behind this view.
    ///
    /// Use this modifier to place one or
    more views behind another view.
    /// For example, you can place a
    collection of stars behind a ``Text``
    view:
    ///
    ///     Text("ABCDEF")
    ///         .background(alignment: .l
    eading) { Star(color: .red) }
    ///         .background(alignment: .c
    enter) { Star(color: .green) }
    ///         .background(alignment: .t
    railing) { Star(color: .blue) }
    ///
    /// The example above assumes that
    you've defined a `Star` view with a
    /// parameterized color:
    ///
    ///     struct Star: View {
    ///         var color: Color
```

```
///  
///         var body: some View {  
///             Image(systemName:  
"star.fill")  
///                 .foregroundStyle(  
color)  
///             }  
///         }  
///  
/// By setting different `alignment`  
values for each modifier, you make the  
/// stars appear in different places  
behind the text:  
///  
///     ! [A screenshot of the letters A,  
B, C, D, E, and F written in front of  
/// three stars. The stars, from left  
to right, are red, green, and  
/// blue.] (View-background-2)  
///  
/// If you specify more than one view  
in the `content` closure, the modifier  
/// collects all of the views in the  
closure into an implicit ``ZStack``,  
/// taking them in order from back to  
front. For example, you can layer a  
/// vertical bar behind a circle,  
with both of those behind a horizontal  
/// bar:  
///  
///     Color.blue  
///         .frame(width: 200,  
height: 10) // Creates a horizontal bar.
```

```
///         .background {
///             Color.green
///                 .frame(width: 10,
height: 100) // Creates a vertical bar.
///                 Circle()
///                 .frame(width: 50,
height: 50)
///             }
///
/// Both the background modifier and
the implicit ``ZStack`` composed from
/// the background content --- the
circle and the vertical bar --- use a
/// default ``Alignment/center``
alignment. The vertical bar appears
/// centered behind the circle, and
both appear as a composite view centered
/// behind the horizontal bar:
///
/// ! [A screenshot of a circle with a
horizontal blue bar layered on top
/// and a vertical green bar layered
underneath. All of the items are center
/// aligned.] (View-background-3)
///
/// If you specify an alignment for
the background, it applies to the
/// implicit stack rather than to the
individual views in the closure. You
/// can see this if you add the
``Alignment/leading`` alignment:
///
///     Color.blue
```

```
    /// .frame(width: 200,
height: 10)
    /// .background(alignment: .l
eading) {
    ///     Color.green
    ///         .frame(width: 10,
height: 100)
    ///             Circle()
    ///                 .frame(width: 50,
height: 50)
    ///             }
    ///
    /// The vertical bar and the circle
move as a unit to align the stack
    /// with the leading edge of the
horizontal bar, while the
    /// vertical bar remains centered on
the circle:
    ///
    /// ! [A screenshot of a horizontal
blue bar in front of a circle, which
    /// is in front of a vertical green
bar. The horizontal bar and the circle
    /// are center aligned with each
other; the left edges of the circle
    /// and the horizontal are aligned.]
```

(View–background–3a)

```
    ///
    /// To control the placement of
individual items inside the `content`-
    /// closure, either use a different
background modifier for each item, as
    /// the earlier example of stars
```

under text demonstrates, or add an explicit

```
    /// ``ZStack`` inside the content
closure with its own alignment:
    ///
    ///     Color.blue
    ///         .frame(width: 200,
height: 10)
    ///             .background(alignment: .l
eading) {
    ///
ZStack(alignment: .leading) {
    ///
        Color.green
    ///
        .frame(width:
10, height: 100)
    ///
        Circle()
    ///
        .frame(width:
50, height: 50)
    ///
    ///         }
    ///
    ///     }
    ///
    /// The stack alignment ensures that
the circle's leading edge aligns with
    /// the vertical bar's, while the
background modifier aligns the composite
    /// view with the horizontal bar:
    ///
    /// ! [A screenshot of a horizontal
blue bar in front of a circle, which
    /// is in front of a vertical green
bar. All items are aligned on their
    /// left edges.] (View-background-4)
    ///
```

```
    /// You can achieve layering without
    a background modifier by putting both
    /// the modified view and the
    background content into a ``ZStack``.
This
    /// produces a simpler view
hierarchy, but it changes the layout
priority
    /// that SwiftUI applies to the
views. Use the background modifier when
you
    /// want the modified view to
dominate the layout.
    ///
    /// If you want to specify a
``ShapeStyle`` like a
    /// ``HierarchicalShapeStyle`` or a
``Material`` as the background, use
    ///
``View/background(_:ignoresSafeAreaEdges:
)` instead.
    /// To specify a ``Shape`` or
``InsettableShape``, use
    ///
``View/background(_:in:fillStyle:)``.
    /// To configure the background of a
presentation, like a sheet, use
    ///
``View/presentationBackground(alignment:c
ontent:)``.
    ///
    /// - Parameters:
    ///   - alignment: The alignment that
```

the modifier uses to position the  
    ///     implicit ``ZStack`` that  
groups the background views. The default  
    ///     is ``Alignment/center``.  
    /// - content: A ``ViewBuilder``  
that you use to declare the views to draw  
    ///     behind this view, stacked in  
a cascading order from bottom to top.  
    ///     The last view that you list  
appears at the front of the stack.  
    ///  
    /// - Returns: A view that uses the  
specified content as a background.  
    @inlinable nonisolated public func  
background<V>(alignment: Alignment  
= .center, @ViewBuilder content: () -> V)  
-> some View where V : View

    /// Sets the view's background to the  
default background style.  
    ///  
    /// This modifier behaves like  
``View/background(\_:ignoresSafeAreaEdges:  
)``,  
    /// except that it always uses the  
``ShapeStyle/background`` shape style.  
    /// For example, you can add a  
background to a ``Label``:  
    ///  
    ///     ZStack {  
    ///         Color.teal  
    ///         Label("Flag",

```
systemImage: "flag.fill")
    /**
     * padding()
     * background()
    }
}

/// Without the background modifier,
the teal color behind the label shows
/// through the label. With the
modifier, the label's text and icon
appear
    /// backed by a region filled with a
color that's appropriate for light
    /// or dark appearance:
    ///
    /// ![A screenshot of a flag icon and
the word flag inside a rectangle; the
    /// rectangle is filled with the
background color and layered on top of a
    /// larger rectangle that's filled
with the color teal.](View-background-7)
    ///
    /// If you want to specify a ``View``
or a stack of views as the background,
    /// use
``View/background(alignment:content:)``
instead.
    /// To specify a ``Shape`` or
``InsettableShape``, use
    ///
``View/background(_:in:fillStyle:)``.
    /// To configure the background of a
presentation, like a sheet, use
    ///
```

```
``View/presentationBackground(_:)``.  
///  
/// - Parameters:  
///   - edges: The set of edges for  
which to ignore safe area insets  
///   when adding the background.  
The default value is ``Edge/Set/all``.  
///   Specify an empty set to  
respect safe area insets on all edges.  
///  
/// - Returns: A view with the  
``ShapeStyle/background`` shape style  
/// drawn behind it.  
@inlinable nonisolated public func  
background(ignoresSafeAreaEdges: Edge.Set = .all) -> some View
```

```
    /// Sets the view's background to a  
style.  
    ///  
    /// Use this modifier to place a type  
that conforms to the ``ShapeStyle``  
    /// protocol --- like a ``Color``,  
``Material``, or  
    /// ``HierarchicalShapeStyle`` ---  
behind a view. For example, you can add  
    /// the  
``ShapeStyle/regularMaterial`` behind a  
``Label``:  
    ///  
    ///     struct FlagLabel: View {  
    ///         var body: some View {
```

```
    /// Label("Flag",
systemImage: "flag.fill")
    /// .padding()
    /// .background(.regu
larMaterial)
    /// }
    /// }
    ///
    /// SwiftUI anchors the style to the
view's bounds. For the example above,
    /// the background fills the entirety
of the label's frame, which includes
    /// the padding:
    ///
    /// ! [A screenshot of a flag symbol
and the word flag layered over a
    /// gray rectangle.] (View-
background-5)
    ///
    /// SwiftUI limits the background
style's extent to the modified view's
    /// container-relative shape. You can
see this effect if you constrain the
    /// `FlagLabel` view with a
``View/containerShape(_:)`` modifier:
    ///
    /// FlagLabel()
    /// .containerShape(RoundedRe
ctangle(cornerRadius: 16))
    ///
    /// The background takes on the
specified container shape:
    ///
```

```
    /// ! [A screenshot of a flag symbol  
and the word flag layered over a  
    /// gray rectangle with rounded  
corners.] (View-background-6)  
    ///  
    /// By default, the background  
ignores safe area insets on all edges,  
but  
    /// you can provide a specific set of  
edges to ignore, or an empty set to  
    /// respect safe area insets on all  
edges:  
    ///  
    ///     Rectangle()  
    ///         .background(  
    ///             .regularMaterial,  
    ///             ignoresSafeAreaEdges:  
[]) // Ignore no safe area insets.  
    ///  
    /// If you want to specify a ``View``  
or a stack of views as the background,  
    /// use  
``View/background(alignment:content:)``  
instead.  
    /// To specify a ``Shape`` or  
``InsettableShape``, use  
    ///  
``View/background(_:in:fillStyle:)`` .  
    /// To configure the background of a  
presentation, like a sheet, use  
    ///  
``View/presentationBackground(_:)``.  
    ///
```

```
    /// - Parameters:  
    ///   - style: An instance of a type  
    ///     that conforms to ``ShapeStyle`` that  
    ///       SwiftUI draws behind the  
    ///     modified view.  
    ///   - edges: The set of edges for  
    ///     which to ignore safe area insets  
    ///       when adding the background.  
    ///     The default value is ``Edge/Set/all``.  
    ///   - Specify an empty set to  
    ///     respect safe area insets on all edges.  
    ///  
    /// - Returns: A view with the  
    ///   specified style drawn behind it.  
    @inlinable nonisolated public func  
background<S>(_ style: S,  
ignoresSafeAreaEdges edges: Edge.Set  
= .all) -> some View where S : ShapeStyle
```

```
    /// Sets the view's background to a  
shape filled with the  
    /// default background style.  
    ///  
    /// This modifier behaves like  
``View/background(_:in:fillStyle:)``,  
    /// except that it always uses the  
``ShapeStyle/background`` shape style  
    /// to fill the specified shape. For  
example, you can create a ``Path``  
    /// that outlines a trapezoid:  
    ///  
    ///     let trapezoid = Path { path
```

```
in
    /**
     *      path.move(to: .zero)
     *      path.addLine(to:
CGPoint(x: 90, y: 0))
     *      path.addLine(to:
CGPoint(x: 80, y: 50))
     *      path.addLine(to:
CGPoint(x: 10, y: 50))
    /**
    /**
    /**
     * Then you can use that shape as a
background for a ``Label``:
    /**
    /**
        ZStack {
    /**
        Color.teal
    /**
        Label("Flag",
systemImage: "flag.fill")
    /**
        .padding()
    /**
        .background(in:
trapezoid)
    /**
    /**
    /**
     * Without the background modifier,
the fill color shows
    /**
        through the label. With the
modifier, the label's text and icon
appear
    /**
        backed by a shape filled with a
color that's appropriate for light
    /**
        or dark appearance:
    /**
    /**
        ! [A screenshot of a flag icon and
the word flag inside a trapezoid; the
```

```
    /// trapezoid is filled with the
background color and layered on top of
    /// a rectangle filled with the color
teal.](View-foreground-B)
    ///
    /// To create a background with other
``View`` types --- or with a stack
    /// of views --- use
``View/background(alignment:content:)`` instead.
    /// To add a ``ShapeStyle`` as a
background, use
    ///
``View/background(_:ignoresSafeAreaEdges:
)``.
    ///
    /// - Parameters:
    ///   - shape: An instance of a type
that conforms to ``Shape`` that
    ///   SwiftUI draws behind the view
using the ``ShapeStyle/background``
    ///   shape style.
    ///   - fillStyle: The ``FillStyle``
to use when drawing the shape.
    ///   The default style uses the
nonzero winding number rule and
    ///   antialiasing.
    ///
    /// - Returns: A view with the
specified shape drawn behind it.
@inlinable nonisolated public func
background<S>(in shape: S, fillStyle:
FillStyle = FillStyle()) -> some View
```

where `S : Shape`

```
    /// Sets the view's background to a
shape filled with a style.
    ///
    /// Use this modifier to layer a type
that conforms to the ``Shape``
    /// protocol behind a view. Specify
the ``ShapeStyle`` that's used to
    /// fill the shape. For example, you
can create a ``Path`` that outlines
    /// a trapezoid:
    ///
    ///     let trapezoid = Path { path
in
    ///         path.move(to: .zero)
    ///         path.addLine(to:
CGPoint(x: 90, y: 0))
    ///         path.addLine(to:
CGPoint(x: 80, y: 50))
    ///         path.addLine(to:
CGPoint(x: 10, y: 50))
    ///     }
    ///
    /// Then you can use that shape as a
background for a ``Label``:
    ///
    ///     Label("Flag", systemImage:
"flag.fill")
    ///         .padding()
    ///         .background(.teal, in:
trapezoid)
```

```
///  
/// The ``ShapeStyle/teal`` color  
fills the shape:  
///  
/// ! [A screenshot of the flag icon  
and the word flag inside a trapezoid;  
/// The trapezoid is filled with the  
color teal.] (View-background-A)  
///  
/// This modifier is a convenience  
method for placing a single shape behind  
a view. To  
/// create a background with other  
``View`` types --- or with a stack  
/// of views --- use  
``View/background(alignment:content:)``  
instead.  
/// To add a ``ShapeStyle`` as a  
background, use  
///  
``View/background(_:ignoresSafeAreaEdges:  
)`.  
///  
/// - Parameters:  
///   - style: A ``ShapeStyle`` that  
SwiftUI uses to fill the shape  
///     that you specify.  
///   - shape: An instance of a type  
that conforms to ``Shape`` that  
///     SwiftUI draws behind the  
view.  
///   - fillStyle: The ``FillStyle``  
to use when drawing the shape.
```

```
    ///      The default style uses the
    nonzero winding number rule and
    ///      antialiasing.
    ///
    /// - Returns: A view with the
    specified shape drawn behind it.
    @inlinable nonisolated public func
background<S, T>(_ style: S, in shape: T,
fillStyle: FillStyle = FillStyle()) ->
some View where S : ShapeStyle, T : Shape
```

```
    /// Sets the view's background to an
    insettable shape filled with the
    /// default background style.
    ///
    /// This modifier behaves like
``View/background(_:in:fillStyle:)``,
    /// except that it always uses the
``ShapeStyle/background`` shape style
    /// to fill the specified insettable
shape. For example, you can use
    /// a ``RoundedRectangle`` as a
background on a ``Label``:
    ///
    ///      ZStack {
    ///          Color.teal
    ///          Label("Flag",
systemImage: "flag.fill")
    ///              .padding()
    ///              .background(in:
RoundedRectangle(cornerRadius: 8))
    ///      }
```

```
///  
/// Without the background modifier,  
the fill color shows  
    /// through the label. With the  
modifier, the label's text and icon  
appear  
    /// backed by a shape filled with a  
color that's appropriate for light  
    /// or dark appearance:  
    ///  
    /// ![A screenshot of a flag icon and  
the word flag inside a rectangle with  
    /// rounded corners; the rectangle is  
filled with the background color, and  
    /// is layered on top of a larger  
rectangle that's filled with the color  
    /// teal.] (View-background-9)  
    ///  
    /// To create a background with other  
``View`` types --- or with a stack  
    /// of views --- use  
``View/background(alignment:content:)``  
instead.  
    /// To add a ``ShapeStyle`` as a  
background, use  
    ///  
``View/background(_:ignoresSafeAreaEdges:  
)` `.  
    ///  
    /// - Parameters:  
    ///     - shape: An instance of a type  
that conforms to ``InsettableShape``  
    ///     that SwiftUI draws behind the
```

```
view using the
    /// ``ShapeStyle/background``  
shape style.
    /// - fillStyle: The ``FillStyle``  
to use when drawing the shape.
    /// The default style uses the  
nonzero winding number rule and
    /// antialiasing.
    ///
    /// - Returns: A view with the  
specified insettable shape drawn behind  
it.
@inlinable nonisolated public func
background<S>(in shape: S, fillStyle:  
FillStyle = FillStyle()) -> some View
where S : InsettableShape
```

```
    /// Sets the view's background to an  
insettable shape filled with a style.
    ///
    /// Use this modifier to layer a type  
that conforms to the
    /// ``InsettableShape`` protocol ---  
like a ``Rectangle``, ``Circle``, or
    /// ``Capsule`` --- behind a view.  
Specify the ``ShapeStyle`` that's used to
    /// fill the shape. For example, you
can place a ``RoundedRectangle``
    /// behind a ``Label``:
    ///
    ///     Label("Flag", systemImage:
"flag.fill")
```

```
    /// .padding()
    /// .background(.teal, in:
RoundedRectangle(cornerRadius: 8))
    ///
    /// The ``ShapeStyle/teal`` color
fills the shape:
    ///
    /// ! [A screenshot of the flag icon
and word on a teal rectangle with
    /// rounded corners.] (View-
background-8)
    ///
    /// This modifier is a convenience
method for placing a single shape behind
a view. To
    /// create a background with other
``View`` types --- or with a stack
    /// of views --- use
``View/background(alignment:content:)``
instead.
    /// To add a ``ShapeStyle`` as a
background, use
    ///
``View/background(_:ignoresSafeAreaEdges:
)``.
    ///
    /// - Parameters:
    ///   - style: A ``ShapeStyle`` that
SwiftUI uses to fill the shape
    ///   that you specify.
    ///   - shape: An instance of a type
that conforms to ``InsettableShape``
    ///   that SwiftUI draws behind the
```

```
view.
```

```
    /// - fillStyle: The ``FillStyle``  
to use when drawing the shape.
```

```
    /// The default style uses the  
nonzero winding number rule and
```

```
    /// antialiasing.
```

```
    ///
```

```
    /// - Returns: A view with the  
specified insettable shape drawn behind  
it.
```

```
    @inlinable nonisolated public func  
background<S, T>(_ style: S, in shape: T,  
fillStyle: FillStyle = FillStyle()) ->  
some View where S : ShapeStyle, T :  
InsettableShape
```

```
}
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension View {
```

```
    /// Modifies the view to use a given  
transition as its method of animating
```

```
    /// changes to the contents of its  
views.
```

```
    ///
```

```
    /// This modifier allows you to  
perform a transition that animates a  
change
```

```
    /// within a single view. The  
provided ``ContentTransition`` can  
present an
```

```
    /// opacity animation for content
    changes, an interpolated animation of
        /// the content's paths as they
    change, or perform no animation at all.
    ///
    /// > Tip: The
`contentTransition(_:)` modifier only has
an effect within
    /// the context of an ``Animation``.
    ///
    /// In the following example, a
``Button`` changes the color and font
size
    /// of a ``Text`` view. Since both of
these properties apply to the paths of
    /// the text, the
``ContentTransition/interpolate``
transition can animate a
    /// gradual change to these
properties through the entire transition.
By
    /// contrast, the
``ContentTransition/opacity`` transition
would simply fade
    /// between the start and end states.
    ///
    ///     private static let font1 =
Font.system(size: 20)
    ///     private static let font2 =
Font.system(size: 45)
    ///
    ///     @State private var color =
Color.red
```

```
    ///      @State private var
currentFont = font1
    /**
     /**
     var body: some View {
     /**
         VStack {
     /**
         Text("Content
transition")
    /**
        .foregroundColor(
color)
    /**
        .font(currentFont
)
    /**
        .contentTransitio
n(.interpolate)
    /**
        Spacer()
    /**
        Button("Change") {
    /**
            withAnimation(Animation.easeInOut(duratio
n: 5.0)) {
    /**
                    color =
(color == .red) ? .green : .red
    /**
                    currentFont =
(currentFont == font1) ? font2 : font1
    /**
                    }
    /**
                    }
    /**
                    }
    /**
                    }
    /**
                    }

    /**
        This example uses an ease-in-
ease-out animation with a five-second
    /**
        duration to make it easier to see
the effect of the interpolation. The
    /**
        figure below shows the `Text` at
the beginning of the animation,
```

```
    /// halfway through, and at the end.  
    ///  
    /// | Time      | Display |  
    /// | ----- | ----- |  
    /// | Start    | ! [The text Content  
transition in a small red font.]  
(ContentTransition-1) |  
    /// | Middle   | ! [The text Content  
transition in a medium brown font.]  
(ContentTransition-2) |  
    /// | End      | ! [The text Content  
transition in a large green font.]  
(ContentTransition-3) |  
    ///  
    /// To control whether content  
transitions use GPU-accelerated  
rendering,  
    /// set the value of the  
    ///  
``EnvironmentValues/contentTransitionAdds  
DrawingGroup`` environment  
    /// variable.  
    ///  
    /// - parameter transition: The  
transition to apply when animating the  
    /// content change.  
    nonisolated public func  
contentTransition(_ transition:  
ContentTransition) -> some View  
  
}  
  
@available(iOS 17.0, macOS 14.0, tvOS
```

```
17.0, watchOS 10.0, *)
extension View {
    /// Applies effects to this view,
    // while providing access to layout
    /// information through a geometry
    proxy.
    ///
    /// You return new effects by calling
    functions on the first argument
    /// provided to the `effect` closure.
    In this example, `ContentView` is
    /// offset by its own size, causing
    its top left corner to appear where the
    /// bottom right corner was
    originally located:
    /// ````swift
    /// ContentView()
    ///     .visualEffect { content,
    geometryProxy in
    ///
    content.offset(geometryProxy.size)
    ///     }
    /// ```
    ///
    /// - Parameters:
    ///     - effect: A closure that
    returns the effect to be applied. The
    first
    ///     argument provided to the
    closure is a placeholder representing
    ///     this view. The second
    argument is a `GeometryProxy`.
```

```
    /// - Returns: A view with the effect applied.
    nonisolated public func visualEffect(_ effect: @escaping @Sendable (EmptyVisualEffect, GeometryProxy) -> some VisualEffect) -> some View
}

@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension View {

    /// Adjusts the color saturation of this view.
    ///
    /// Use color saturation to increase or decrease the intensity of colors in a view.
    ///
    /// The example below shows a series of red squares with their saturation increasing from 0 (gray) to 100% (fully-red) in 20% increments:
    ///
    ///     struct Saturation: View {
    ///         var body: some View {
    ///             HStack {
    ///                 ForEach(0..<6) {
    ///
    /// Color.red.frame(width: 60, height: 60,
    alignment: .center)
```



```
13.0, watchOS 6.0, *)
extension View {

    /// Changes the view's proposed area
    /// to extend outside the screen's safe
    /// areas.
    ///
    /// Use `edgesIgnoringSafeArea(_:)` to
    /// change the area proposed for this
    /// view so that – were the proposal
    /// accepted – this view could extend
    /// outside the safe area to the
    /// bounds of the screen for the specified
    /// edges.
    ///
    /// For example, you can propose that
    /// a text view ignore the safe area's top
    /// inset:
    ///
    ///     VStack {
    ///         Text("This text is
    /// outside of the top safe area.")
    ///         .edgesIgnoringSafeAre
    /// a([.top])
    ///         .border(Color.purple)
    ///         Text("This text is inside
    /// VStack.")
    ///         .border(Color.yellow)
    ///     }
    ///     .border(Color.gray)
    ///
    /// ! [A screenshot showing a view
    /// whose bounds exceed the safe area of the
```

```
    /// screen.](SwiftUI-View-
edgesIgnoringSafeArea.png)
    /**
     /// Depending on the surrounding view
hierarchy, SwiftUI may not honor an
     /// `edgesIgnoringSafeArea(_:)`  
request. This can happen, for example, if
     /// the view is inside a container
that respects the screen's safe area. In
     /// that case you may need to apply
`edgesIgnoringSafeArea(_:)` to the
     /// container instead.
    /**
     /// - Parameter edges: The set of the
edges in which to expand the size
     /// requested for this view.
    /**
     /// - Returns: A view that may extend
outside of the screen's safe area
     /// on the edges specified by
`edges`.
    @available(iOS, introduced: 13.0,
deprecated: 100000.0, message: "Use
ignoresSafeArea(_:edges:) instead.")
    @available(macOS, introduced: 10.15,
deprecated: 100000.0, message: "Use
ignoresSafeArea(_:edges:) instead.")
    @available(tvOS, introduced: 13.0,
deprecated: 100000.0, message: "Use
ignoresSafeArea(_:edges:) instead.")
    @available(watchOS, introduced: 6.0,
deprecated: 100000.0, message: "Use
ignoresSafeArea(_:edges:) instead.")
```

```
    @available(visionOS, introduced: 1.0,
deprecated: 100000.0, message: "Use
ignoresSafeArea(_:edges:) instead.")
    @inlinable nonisolated public func
edgesIgnoringSafeArea(_ edges: Edge.Set)
-> some View

}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
extension View {

    /// Expands the safe area of a view.
    ///
    /// By default, the SwiftUI layout
    /// system sizes and positions views to
    /// avoid certain safe areas. This
    /// ensures that system content like the
    /// software keyboard or edges of the
    /// device don't obstruct your
    /// views. To extend your content
    /// into these regions, you can ignore
    /// safe areas on specific edges by
    /// applying this modifier.
    ///
    /// For examples of how to use this
    /// modifier,
    /// see <doc:Adding-a-Background-to-
    /// Your-View>.
    ///
    /// - Parameters:
    ///   - regions: The regions to
```

```
expand the view's safe area into. The
    /// modifier expands into all
safe area region types by default.
    /// - edges: The set of edges to
expand. Any edges that you
    /// don't include in this set
remain unchanged. The set includes all
    /// edges by default.
    ///
    /// - Returns: A view with an
expanded safe area.
@inlinable nonisolated public func
ignoresSafeArea(_ regions:
SafeAreaRegions = .all, edges: Edge.Set =
.all) -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Scales images within the view
according to one of the relative sizes
    /// available including small,
medium, and large images sizes.
    ///
    /// The example below shows the
relative scaling effect. The system
renders
    /// the image at a relative size
based on the available space and
    /// configuration options of the
```

```
image it is scaling.  
    ///  
    ///      VStack {  
    ///          HStack {  
    ///              Image(systemName:  
"heart.fill")  
    ///                  .imageScale(.smal  
l)  
    ///          Text("Small")  
    ///      }  
    ///      HStack {  
    ///          Image(systemName:  
"heart.fill")  
    ///                  .imageScale(.medi  
um)  
    ///          Text("Medium")  
    ///      }  
    ///      HStack {  
    ///          Image(systemName:  
"heart.fill")  
    ///                  .imageScale(.larg  
e)  
    ///          Text("Large")  
    ///      }  
    ///  }  
    /// ! [A view showing small, medium,  
and large hearts rendered at a size  
    /// relative to the available space.]  
(SwiftUI-View-imageScale.png)  
    ///  
    /// - Parameter scale: One of the
```

```
relative sizes provided by the image
scale
    /// enumeration.
    @available(macOS 11.0, *)
    @inlinable nonisolated public func
imageScale(_ scale: Image.Scale) -> some
View

    /// Sets the default font for text in
this view.
    ///
    /// Use `font(_:)` to apply a
specific font to all of the text in a
view.
    ///
    /// The example below shows the
effects of applying fonts to individual
    /// views and to view hierarchies.
Font information flows down the view
    /// hierarchy as part of the
environment, and remains in effect unless
    /// overridden at the level of an
individual view or view container.
    ///
    /// Here, the outermost ``VStack``
applies a 16-point system font as a
    /// default font to views contained
in that ``VStack``. Inside that stack,
    /// the example applies a
``Font/largeTitle`` font to just the
first text
    /// view; this explicitly overrides
```

the default. The remaining stack and the  
    /// views contained with it continue  
to use the 16-point system font set by  
    /// their containing view:  
    ///  
    ///       VStack {  
    ///           Text("Font applied to a  
text view.")  
    ///            .font(.largeTitle)  
    ///  
    ///       VStack {  
    ///           Text("These 2 text  
views have the same font")  
    ///           Text("applied to  
their parent hierarchy")  
    ///           }  
    ///       }  
    ///       .font(.system(size: 16,  
weight: .light, design: .default))  
    ///  
    /// ! [A screenshot showing the  
application fonts to an individual text  
field  
    /// and view hierarchy.] (SwiftUI-  
view-font.png)  
    ///  
    /// - Parameter font: The default  
font to use in this view.  
    ///  
    /// - Returns: A view with the  
default font set to the value you supply.  
    @inlinable nonisolated public func  
font(\_ font: Font?) -> some View

```
    /// Modifies the fonts of all child
views to use fixed-width digits, if
    /// possible, while leaving other
characters proportionally spaced.
    ///
    /// Using fixed-width digits allows
you to easily align numbers of the
    /// same size in a table-like
arrangement. This feature is also known
as
    /// "tabular figures" or "tabular
numbers."
    ///
    /// This modifier only affects
numeric characters, and leaves all other
    /// characters unchanged.
    ///
    /// The following example shows the
effect of `monospacedDigit()` on
    /// multiple child views. The example
consists of two ``VStack`` views
    /// inside an ``HStack``. Each
`VStack` contains two ``Button`` views,
with
    /// the second `VStack` applying the
`monospacedDigit()` modifier to its
    /// contents. As a result, the digits
in the buttons in the trailing
    /// `VStack` are the same width,
which in turn gives the buttons equal
widths.
```

```
///  
///     var body: some View {  
///         HStack(alignment: .top) {  
///  
VStack(alignment: .leading) {  
    ///             Button("Delete  
111 messages") {}  
    ///             Button("Delete  
222 messages") {}  
    ///         }  
    ///  
VStack(alignment: .leading) {  
    ///             Button("Delete  
111 messages") {}  
    ///             Button("Delete  
222 messages") {}  
    ///         }  
    ///         .monospacedDigit()  
    ///         .padding()  
    ///         .navigationTitle("monospa  
cedDigit() Child Views")  
    ///     }  
    ///  
    ///     ! [A macOS window showing four  
buttons, arranged in two columns. Each  
    /// column's buttons contain the same  
text: Delete 111 messages and Delete  
    /// 222 messages. The right column's  
buttons have fixed width, or  
    /// monospaced, digits, which make  
the 1 characters wider than they would be  
    /// in a proportional font. Because
```

```
the 1 and 2 characters are the same
    /// width in the right column, the
buttons are the same
    /// width.] (View-monospacedDigit-1)
    ///
    /// If a child view's base font
doesn't support fixed-width digits, the
font
    /// remains unchanged.
    ///
    /// - Returns: A view whose child
views' fonts use fixed-width numeric
    /// characters, while leaving other
characters proportionally spaced.
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
nonisolated public func
monospacedDigit() -> some View

    /// Modifies the fonts of all child
views to use the fixed-width variant of
    /// the current font, if possible.
    ///
    /// If a child view's base font
doesn't support fixed-width, the font
    /// remains unchanged.
    ///
    /// - Returns: A view whose child
views' fonts use fixed-width characters,
    /// while leaving other characters
proportionally spaced.
@available(iOS 16.0, macOS 13.0, tvOS
```

```
16.0, watchOS 9.0, *)
    nonisolated public func monospaced(_
isActive: Bool = true) -> some View
```

```
    /// Sets the font weight of the text
in this view.
```

```
    ///
```

```
    /// - Parameter weight: One of the
available font weights.
```

```
    /// Providing `nil` removes the
effect of any font weight
```

```
    /// modifier applied higher in the
view hierarchy.
```

```
    ///
```

```
    /// - Returns: A view that uses the
font weight you specify.
```

```
@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
```

```
    nonisolated public func fontWeight(_
weight: Font.Weight?) -> some View
```

```
    /// Sets the font width of the text
in this view.
```

```
    ///
```

```
    /// - Parameter width: One of the
available font widths.
```

```
    /// Providing `nil` removes the
effect of any font width
```

```
    /// modifier applied higher in the
view hierarchy.
```

```
    ///
```

```
    /// - Returns: A view that uses the  
    font width you specify.  
    @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
    nonisolated public func fontWidth(_  
width: Font.Width?) -> some View
```

```
    /// Applies a bold font weight to the  
text in this view.
```

```
///
```

```
    /// - Parameter isActive: A Boolean  
value that indicates
```

```
        /// whether bold font styling is  
added. The default value is `true`.
```

```
///
```

```
    /// - Returns: A view with bold text.
```

```
    @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)
```

```
    nonisolated public func bold(_  
isActive: Bool = true) -> some View
```

```
    /// Applies italics to the text in  
this view.
```

```
///
```

```
    /// - Parameter isActive: A Boolean  
value that indicates
```

```
        /// whether italic styling is  
added. The default value is `true`.
```

```
///
```

```
    /// - Returns: A View with italic  
text.
```

```
    @available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
    nonisolated public func italic(_  
isActive: Bool = true) -> some View
```

/// Sets the font design of the text  
in this view.

///

/// – Parameter design: One of the  
available font designs.

/// Providing `nil` removes the  
effect of any font design

/// modifier applied higher in the  
view hierarchy.

///

/// – Returns: A view that uses the  
font design you specify.

```
    @available(iOS 16.1, macOS 13.0, tvOS  
16.1, watchOS 9.1, *)
```

```
    nonisolated public func fontDesign(_  
design: Font.Design?) -> some View
```

/// Sets the spacing, or kerning,  
between characters for the text in this  
view.

///

/// – Parameter kerning: The spacing  
to use between individual characters in  
text.

/// Value of `0` sets the kerning  
to the system default value.

```
///  
/// - Returns: A view where text has  
the specified amount of kerning.  
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
nonisolated public func kerning(_  
kerning: CGFloat) -> some View
```

```
/// Sets the tracking for the text in  
this view.
```

```
///  
/// - Parameter tracking: The amount  
of additional space, in points, that  
/// the view should add to each  
character cluster after layout. Value of  
'0'
```

```
/// sets the tracking to the system  
default value.
```

```
///  
/// - Returns: A view where text has  
the specified amount of tracking.
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
nonisolated public func tracking(_  
tracking: CGFloat) -> some View
```

```
/// Sets the vertical offset for the  
text relative to its baseline  
/// in this view.
```

```
///  
/// - Parameter baselineOffset: The
```

```
amount to shift the text
    /// vertically (up or down)
relative to its baseline.
    ///
    /// - Returns: A view where text is
above or below its baseline.
    @available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
    nonisolated public func
baselineOffset(_ baselineOffset: CGFloat)
-> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Adds a color multiplication
effect to this view.
    ///
    /// The following example shows two
versions of the same image side by side;
    /// at left is the original, and at
right is a duplicate with the
    /// `colorMultiply(_:)` modifier
applied with ``ShapeStyle/purple``.
    ///
    ///     struct InnerCircleView: View
{
    ///         var body: some View {
    ///             Circle()
    ///                 .fill(Color.green
```

```
)  
    /// .frame(width: 40,  
height: 40, alignment: .center)  
    /// }  
    /// }  
    ///  
    /// struct ColorMultiply: View {  
    ///     var body: some View {  
    ///         HStack {  
    ///             Color.red.frame(width: 100, height: 100,  
alignment: .center)  
                /// .overlay(Inne  
rCircleView(), alignment: .center)  
                /// .overlay(Text  
("Normal"))  
                ///  
.font(.callout),  
                ///  
alignment: .bottom)  
                /// .border(Color  
.gray)  
                ///  
                /// Spacer()  
                ///  
                ///  
                ///  
Color.red.frame(width: 100, height: 100,  
alignment: .center)  
    /// .overlay(Inne  
rCircleView(), alignment: .center)  
    /// .colorMultipl  
y(Color.purple)  
    /// .overlay(Text
```



```
preference key, the value is a
    /// function of the key's current
value and a geometry value tied
    /// to the current coordinate space,
allowing readers of the value
    /// to convert the geometry to their
local coordinates.

    ///
    /// - Parameters:
    ///   - key: the preference key type.
    ///   - value: the geometry value in
the current coordinate space.
    ///   - transform: the function to
produce the preference value.

    ///
    /// - Returns: a new version of the
view that writes the preference.

    @inlinable nonisolated public func
transformAnchorPreference<A, K>(key _: K.Type = K.self, value: Anchor<A>.Source,
transform: @escaping (inout K.Value,
Anchor<A>) -> Void) -> some View where
K : PreferenceKey

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Sets a clipping shape for this
view.
    ///

```

```
    /// Use `clipShape(_:style:)` to clip
the view to the provided shape. By
    /// applying a clipping shape to a
view, you preserve the parts of the view
    /// covered by the shape, while
eliminating other parts of the view. The
    /// clipping shape itself isn't
visible.
    ///
    /// For example, this code applies a
circular clipping shape to a `Text`
    /// view:
    ///
    ///     Text("Clipped text in a
circle")
    ///         .frame(width: 175,
height: 100)
    ///         .foregroundColor(Color.wh
ite)
    ///         .background(Color.black)
    ///         .clipShape(Circle())
    ///
    /// The resulting view shows only the
portion of the text that lies within
    /// the bounds of the circle.
    ///
    /// ! [A screenshot of text clipped to
the shape of a
    /// circle.] (SwiftUI-View-
clipShape.png)
    ///
    /// - Parameters:
    ///   - shape: The clipping shape to
```

```
use for this view. The `shape` fills
    ///      the view's frame, while
maintaining its aspect ratio.
    /// - style: The fill style to use
when rasterizing `shape`.
    ///
    /// - Returns: A view that clips this
view to `shape`, using `style` to
    /// define the shape's
rasterization.
@inlinable nonisolated public func
clipShape<S>(_ shape: S, style: FillStyle
= FillStyle()) -> some View where S :
Shape
```

```
    /// Clips this view to its bounding
rectangular frame.
    ///
    /// Use the `clipped(antialiased:)`-
modifier to hide any content that
    /// extends beyond the layout bounds
of the shape.
    ///
    /// By default, a view's bounding
frame is used only for layout, so any
    /// content that extends beyond the
edges of the frame is still visible.
    ///
    ///     Text("This long text string
is clipped")
    ///         .fixedSize()
    ///         .frame(width: 175,
```

```
height: 100)
    /**
     * Clips this view to its bounding frame, with the specified corner radius.
     */
    /**
     * By default, a view's bounding frame only affects its layout, so any content that extends beyond the edges of the frame remains visible. Use `cornerRadius(_:antialiased:)` to hide any content that extends beyond these edges while applying a corner radius.
    */
}

@inlinable nonisolated public func clipped(antialiased: Bool = false) -> some View
```

```
    /**
     * Clipping a view to its frame clips its content to the boundaries of the frame. This is useful for creating views with specific shapes or sizes.
     */
    /**
     * Parameter antialiased: A Boolean value that indicates whether the rendering system applies smoothing to the edges of the clipping rectangle.
     */
    /**
     * Returns: A view that clips this view to its bounding frame.
    */

    @inlinable nonisolated public func clipped(antialiased: Bool = false) -> some View
```

```
    /// The following code applies a
    corner radius of 25 to a text view:
    /**
     * Text("Rounded Corners")
     * .frame(width: 175,
height: 75)
     * .foregroundColor(Color.wh
ite)
     * .background(Color.black)
     * .cornerRadius(25)
     *
     * ! [A screenshot of a rectangle
with rounded corners bounding a text
     // view.](SwiftUI-View-
cornerRadius.png)
     */
     /**
      * - Parameter antialiased: A
Boolean value that indicates whether the
      * rendering system applies
smoothing to the edges of the clipping
      * rectangle.
      */
     /**
      * - Returns: A view that clips this
view to its bounding frame with the
      * specified corner radius.
      */
@available(iOS, introduced: 13.0,
deprecated: 100000.0, message: "Use
`clipShape` or `fill` instead.")
@available(macOS, introduced: 10.15,
deprecated: 100000.0, message: "Use
`clipShape` or `fill` instead.")
@available(tvOS, introduced: 13.0,
deprecated: 100000.0, message: "Use
```

```
`clipShape` or `fill` instead.")
    @available(watchOS, introduced: 6.0,
deprecated: 100000.0, message: "Use
`clipShape` or `fill` instead.")
    @available(visionOS, introduced: 1.0,
deprecated: 100000.0, message: "Use
`clipShape` or `fill` instead.")
    @inlinable nonisolated public func
cornerRadius(_ radius: CGFloat,
antialiased: Bool = true) -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Applies the given transaction
mutation function to all animations used
    /// within the view.
    ///
    /// Use this modifier to change or
replace the animation used in a view.
    /// Consider three identical
animations controlled by a
    /// button that executes all three
animations simultaneously:
    ///
    /// * The first animation rotates
the "Rotation" ``Text`` view by 360
    /// degrees.
    /// * The second uses the
`transaction(_:)` modifier to change the
```

```
    ///      animation by adding a delay to
the start of the animation
    ///      by two seconds and then
increases the rotational speed of the
    ///      "Rotation\nModified" ``Text``
view animation by a factor of 2.
    /// * The third animation uses the
`transaction(_:)` modifier to
    ///      replace the rotation animation
affecting the "Animation\nReplaced"
    ///      ``Text`` view with a spring
animation.
    ///
    /// The following code implements
these animations:
    ///
    ///      struct TransactionExample:
View {
    ///          @State private var flag =
false
    ///
    ///          var body: some View {
    ///              VStack(spacing: 50) {
    ///                  HStack(spacing:
30) {
    ///
    ///          Text("Rotation")
    ///              .rotation
Effect(Angle(degrees:
    ///
self.flag ? 360 : 0))
    ///
    ///
```

```
Text("Rotation\nModified")
    /**
     * rotation
Effect(Angle(degrees:
    /**
self.flag ? 360 : 0))
    /**
     * transact
ion { view in
    /**
view.animation =
    /**
view.animation?.delay(2.0).speed(2)
    /**
}
    /**
    /**
Text("Animation\nReplaced")
    /**
     * rotation
Effect(Angle(degrees:
    /**
self.flag ? 360 : 0))
    /**
     * transact
ion { view in
    /**
view.animation = .interactiveSpring(
    /**
response: 0.60,
    /**
dampingFraction: 0.20,
    /**
blendDuration: 0.25)
    /**
}
    /**
    /**
    /**
     * Button("Animate")
```

```
{  
    ///  
    withAnimation(.easeIn(duration: 2.0)) {  
        ///  
        self.flag.toggle()  
        /// }  
        /// }  
        /// }  
        /// }  
        /// }  
        /// }  
        /// Use this modifier on leaf views  
such as ``Image`` or ``Button`` rather  
than container views such as  
``VStack`` or ``HStack``. The  
/// transformation applies to all  
child views within this view; calling  
/// `transaction(_:)` on a container  
view can lead to unbounded scope of  
/// execution depending on the depth  
of the view hierarchy.  
    ///  
    /// - Parameter transform: The  
transformation to apply to transactions  
    /// within this view.  
    ///  
    /// - Returns: A view that wraps this  
view and applies a transformation to  
    /// all transactions used within  
the view.  
    @inlinable nonisolated public func  
transaction(_ transform: @escaping (inout  
Transaction) -> Void) -> some View
```

```
    /// Applies the given transaction
mutation function to all animations used
    /// within the view.
    ///
    /// Use this modifier to change or
replace the animation used in a view.
    /// Consider three identical views
controlled by a
    /// button that changes all three
simultaneously:
    ///
    /// * The first view animates
rotating the "Rotation" ``Text`` view by
360
    /// degrees.
    /// * The second uses the
`transaction(_:)` modifier to change the
    /// animation by adding a delay to
the start of the animation
    /// by two seconds and then
increases the rotational speed of the
    /// "Rotation\nModified" ``Text``
view animation by a factor of 2.
    /// * The third uses the
`transaction(_:)` modifier to disable
animations
    /// affecting the
"Animation\nReplaced" ``Text`` view.
    ///
    /// The following code implements
these animations:
```

```
///  
///     struct TransactionExample:  
View {  
    ///         @State var flag = false  
    ///  
    ///             var body: some View {  
    ///                 VStack(spacing: 50) {  
    ///                     HStack(spacing:  
30) {  
    ///                         Text("Rotation")  
    ///                             .rotation  
Effect(Angle(degrees: flag ? 360 : 0))  
    ///  
    ///  
    Text("Rotation\nModified")  
    ///         .rotation  
Effect(Angle(degrees: flag ? 360 : 0))  
    ///         .transact  
ion(value: flag) { t in  
    ///  
t.animation =  
    ///  
t.animation?.delay(2.0).speed(2)  
    ///  
    ///  
    ///  
    Text("Animation\nReplaced")  
    ///         .rotation  
Effect(Angle(degrees: flag ? 360 : 0))  
    ///         .transact  
ion(value: flag) { t in  
    ///
```



```
    /// Applies the given animation to
    all animatable values within this view.
    ///
    /// Use this modifier on leaf views
    rather than container views. The
    /// animation applies to all child
    views within this view; calling
    /// `animation(_:)` on a container
    view can lead to unbounded scope.
    ///
    /// - Parameter animation: The
    animation to apply to animatable values
    /// within this view.
    ///
    /// - Returns: A view that wraps this
    view and applies `animation` to all
    /// animatable values used within
    the view.
    @available(iOS, introduced: 13.0,
    deprecated: 15.0, message: "Use
    withAnimation or animation(_:value:)
    instead.")
    @available(macOS, introduced: 10.15,
    deprecated: 12.0, message: "Use
    withAnimation or animation(_:value:)
    instead.")
    @available(tvOS, introduced: 13.0,
    deprecated: 15.0, message: "Use
    withAnimation or animation(_:value:)
    instead.")
    @available(watchOS, introduced: 6.0,
```

```
deprecated: 8.0, message: "Use
withAnimation or animation(_:value:)
instead.")
    @available(visionOS, introduced: 1.0,
deprecated: 1.0, message: "Use
withAnimation or animation(_:value:)
instead.")
    @inlinable nonisolated public func
animation(_ animation: Animation?) ->
some View

}

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

    /// Applies the given transaction
mutation function to all animations used
    /// within the `body` closure.
    ///
    /// Any modifiers applied to the
content of `body` will be applied to this
    /// view, and the changes to the
transaction performed in the `transform`'
    /// will only affect the modifiers
defined in the `body`.
    ///
    /// The following code animates the
opacity changing with a faster
    /// animation, while the contents of
MyView are animated with the implicit
    /// transaction:
```

```
///  
///      MyView(isActive: isActive)  
///          .transaction  
{ transaction in  
    ///          transaction.animation  
= transaction.animation?.speed(2)  
    ///      } body: { content in  
    ///  
content.opacity(isActive ? 1.0 : 0.0)  
    ///      }  
    ///  
    /// - See Also:  
`Transaction.disablesAnimations`  
nonisolated public func  
transaction<V>(_ transform: @escaping  
(inout Transaction) -> Void, @ViewBuilder  
body: (PlaceholderContentView<Self>) ->  
V) -> some View where V : View
```

```
    /// Applies the given animation to  
all animatable values within the `body`  
    /// closure.  
    ///  
    /// Any modifiers applied to the  
content of `body` will be applied to this  
    /// view, and the `animation` will  
only be used on the modifiers defined in  
    /// the `body`.  
    ///  
    /// The following code animates the  
opacity changing with an easeInOut  
    /// animation, while the contents of
```

```
MyView are animated with the implicit
    /// transaction's animation:
    ///
    ///     MyView(isActive: isActive)
    ///         .animation(.easeInOut)
{ content in
    ///
    content.opacity(isActive ? 1.0 : 0.0)
    ///
}

@nonisolated public func
animation<V>(_ animation: Animation?,
@ViewBuilder body:
(PlaceholderContentView<Self>) -> V) ->
some View where V : View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Binds a view's identity to the
    given proxy value.
    ///
    /// When the proxy value specified by
    the `id` parameter changes, the
    /// identity of the view – for
    example, its state – is reset.
    @inlinable nonisolated public func
id<ID>(_ id: ID) -> some View where ID :
Hashable

}
```

```
@available(iOS, introduced: 13.0,
deprecated: 100000.0, renamed:
"preferredColorScheme(_:)")
@available(macOS, introduced: 10.15,
deprecated: 100000.0, renamed:
"preferredColorScheme(_:)")
@available(tvOS, introduced: 13.0,
deprecated: 100000.0, renamed:
"preferredColorScheme(_:)")
@available(watchOS, introduced: 6.0,
deprecated: 100000.0, renamed:
"preferredColorScheme(_:)")
@available(visionOS, introduced: 1.0,
deprecated: 100000.0, renamed:
"preferredColorScheme(_:)")
extension View {
    /// Sets this view's color scheme.
    ///
    /// Use `colorScheme(_:)` to set the
    /// color scheme for the view to which you
    /// apply it and any subviews. If you
    /// want to set the color scheme for all
    /// views in the presentation, use
    ``View/preferredColorScheme(_:)``
    /// instead.
    ///
    /// - Parameter colorScheme: The
    ///   color scheme for this view.
    ///
    /// - Returns: A view that sets this
    ///   view's color scheme.
```

```
    @inlinable nonisolated public func
colorScheme(_ colorScheme: ColorScheme)
-> some View

}

@available(iOS 13.0, macOS 11.0, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Sets the preferred color scheme
for this presentation.
    ///
    /// Use one of the values in
``ColorScheme`` with this modifier to set
a
    /// preferred color scheme for the
nearest enclosing presentation, like a
    /// popover, a sheet, or a window.
The value that you set overrides the
    /// user's Dark Mode selection for
that presentation. In the example below,
    /// the ``Toggle`` controls an
`isDarkMode` state variable, which in
turn
    /// controls the color scheme of the
sheet that contains the toggle:
    ///
    ///     @State private var
isPresented = false
    ///     @State private var isDarkMode
= true
    ///
```

```
///      var body: some View {
///          Button("Show Sheet") {
///              isPresented = true
///          }
///          .sheet(isPresented:
$.isPresented) {
///              List {
///                  Toggle("Dark
Mode", isOn: $isDarkMode)
///              }
///              .preferredColorScheme
(isDarkMode ? .dark : nil)
///          }
///      }
///      /// If you apply the modifier to any
of the views in the sheet --- which in
/// this case are a ``List`` and a
``Toggle`` --- the value that you set
/// propagates up through the view
hierarchy to the enclosing
/// presentation, or until another
color scheme modifier higher in the
/// hierarchy overrides it. The value
you set also flows down to all child
/// views of the enclosing
presentation.
///
/// Pass `nil` to indicate that there
is no preferred color scheme for this
/// view. This is useful in cases
where a preferred color scheme should
only
```

```
    /// be applied conditionally. In the
    previous example, the sheet will prefer
    /// dark mode when `isDarkMode` is
    set to `true`, but otherwise defer to the
    /// color scheme as determined by the
    system.

    ///
    /// Multiple views in the same view
    hierarchy can set a preferred color
    /// scheme. Applying this modifier
    overrides any existing preferred color
    /// scheme set for the view, such as
    by one of its subviews. If there are
    /// conflicting, non-`nil` color
    scheme preferences set by parallel
    branches
    /// of the view hierarchy, the system
    will prioritize the first non-`nil`
    /// preference based on the order of
    the views. In the example below, the
    /// preferred color scheme for the
    entire view will resolve to ` `.dark` , from
    /// the second view:
    ///
    ///     VStack {
    ///         Text("First")
    ///             .preferredColorScheme
    (.light)
    ///             .preferredColorScheme
    (nil) // overrides ` `.light` 
    ///
    ///         Text("Second")
    ///             .preferredColorScheme
```

```
(.dark)
    /**
     /**
     /**
     Text("Third")
        .preferredColorScheme
(.light)
    /**
    }
    /**
    /**
    A common use for this modifier is
to create side-by-side previews
    /**
    of the same view with light and
dark appearances:
    /**
    /**
    struct MyView_Previews:
PreviewProvider {
    /**
    static var previews: some
View {
    /**
MyView().preferredColorScheme(.light)
    /**
MyView().preferredColorScheme(.dark)
    /**
    }
    /**
    }
    /**
    /**
    If you need to detect the color
scheme that currently applies to a view,
    /**
    read the
``EnvironmentValues/colorScheme``
environment value:
    /**
    /**
    @Environment(\.colorScheme)
private var colorScheme
    /**
    /**
    var body: some View {
```

```
    ///      Text(colorScheme == .dark
? "Dark" : "Light")
    ///
    ///
    /// - Parameter colorScheme: The
    preferred color scheme for this view, or
    /// `nil` to indicate no
    preference.
    ///
    /// - Returns: A view that sets the
    color scheme.
    @inlinable nonisolated public func
preferredColorScheme(_ colorScheme:
ColorScheme?) -> some View

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Attaches a gesture to the view
    with a lower precedence than gestures
    /// defined by the view.
    ///
    /// Use this method when you need to
    attach a gesture to a view. The
    /// example below defines a custom
    gesture that prints a message to the
    /// console and attaches it to the
    view's ``VStack``. Inside the ``VStack``
    /// a red heart ``Image`` defines its
    own ``TapGesture``
```

```
    /// handler that also prints a
message to the console, and blue
rectangle
    /// with no custom gesture handlers.
Tapping or clicking the image
    /// prints a message to the console
from the tap gesture handler on the
    /// image, while tapping or clicking
the rectangle inside the ``VStack``
    /// prints a message in the console
from the enclosing vertical stack
    /// gesture handler.

    ///
    /**
     * struct GestureExample: View {
     *     @State private var
message = "Message"
     *         let newGesture =
TapGesture().onEnded {
     *             print("Tap on
VStack.")
     *         }
     *
     */
     *         var body: some View {
     *             VStack(spacing:25) {
     *                 Image(systemName:
"heart.fill")
     *                     .resizable()
     *                     .frame(width:
75, height: 75)
     *                     .padding()
     *                     .foregroundCo
lor(.red)
     *                     .onTapGesture
```

```
{  
    ///  
    print("Tap on image.")  
    ///  
    ///  
    ///  
    ///  
    }  
    Rectangle()  
        .fill(Color.b  
lue)  
    ///  
    ///  
    ///  
    ///  
    .gesture(newGesture)  
    .frame(width: 200,  
height: 200)  
    ///  
    ///  
    ///  
    .border(Color.purple)  
    ///  
    ///  
    ///  
    ///  
    /// – Parameters:  
    /// – gesture: A gesture to attach  
to the view.  
    /// – mask: A value that controls  
how adding this gesture to the view  
    /// affects other gestures  
recognized by the view and its subviews.  
    /// Defaults to  
``SwiftUI/GestureMask/all``.  
    nonisolated public func gesture<T>(  
gesture: T, including mask: GestureMask =  
.all) -> some View where T : Gesture
```

```
    /// Attaches a gesture to the view  
    with a higher precedence than gestures  
    /// defined by the view.  
    ///
```

```
    /// Use this method when you need to
    define a high priority gesture
    /// to take precedence over the
    view's existing gestures. The
    /// example below defines a custom
    gesture that prints a message to the
    /// console and attaches it to the
    view's ``VStack``. Inside the ``VStack``
    /// a red heart ``Image`` defines its
    own ``TapGesture`` handler that
    /// also prints a message to the
    console, and a blue rectangle
    /// with no custom gesture handlers.
    Tapping or clicking any of the
    /// views results in a console
    message from the high priority gesture
    /// attached to the enclosing
    ``VStack``.

    /**
     * struct
    HighPriorityGestureExample: View {
        /**
         * @State private var
        message = "Message"
        /**
         * let newGesture =
        TapGesture().onEnded {
            /**
             * print("Tap on
        VStack.")
            /**
        }
        /**
         * var body: some View {
        VStack(spacing:25) {
            /**
             * Image(systemName:
        "heart.fill")
```



```
highPriorityGesture<T>(_ gesture: T,  
including mask: GestureMask = .all) ->  
some View where T : Gesture
```

```
    /// Attaches a gesture to the view to  
process simultaneously with gestures  
    /// defined by the view.  
    ///  
    /// Use this method when you need to  
define and process a view specific  
    /// gesture simultaneously with the  
same priority as the  
    /// view's existing gestures. The  
example below defines a custom gesture  
    /// that prints a message to the  
console and attaches it to the view's  
    /// ``VStack``. Inside the ``VStack``  
is a red heart ``Image`` defines its  
    /// own ``TapGesture`` handler that  
also prints a message to the console  
    /// and a blue rectangle with no  
custom gesture handlers.  
    ///  
    /// Tapping or clicking the "heart"  
image sends two messages to the  
    /// console: one for the image's tap  
gesture handler, and the other from a  
    /// custom gesture handler attached  
to the enclosing vertical stack.  
    /// Tapping or clicking on the blue  
rectangle results only in the single  
    /// message to the console from the
```

```
tap recognizer attached to the
    /// ``VStack``:
    ///
    ///     struct
SimultaneousGestureExample: View {
    ///         @State private var
message = "Message"
    ///             let newGesture =
TapGesture().onEnded {
    ///                 print("Gesture on
VStack.")
    ///
    ///
    ///             var body: some View {
    ///                 VStack(spacing:25) {
    ///                     Image(systemName:
"heart.fill")
    ///                         .resizable()
    ///                         .frame(width:
75, height: 75)
    ///                         .padding()
    ///                         .foregroundCo
lor(.red)
    ///                         .onTapGesture
{
    ///
print("Gesture on image.")
    ///
    ///             }
    ///             Rectangle()
    ///                 .fill(Color.b
lue)
    ///
    ///             }
    ///             .simultaneousGesture(
```

```
newGesture)
    /**
     * - Parameters:
     *   - gesture: A gesture to attach
     *     to the view.
     *   - mask: A value that controls
     *     how adding this gesture to the view
     *     affects other gestures
     *     recognized by the view and its subviews.
     */
    @nonisolated public func
    simultaneousGesture<T>(_ gesture: T,
    including mask: GestureMask = .all) ->
    some View where T : Gesture
```

```
    /**
     * Attaches a gesture to the view
     * with a lower precedence than gestures
     * defined by the view.
     */
    /**
     * Use this method when you need to
     * attach a gesture to a view. The
     * example below defines a custom
     * gesture that prints a message to the
     * console and attaches it to the
     * view's ``VStack``. Inside the ``VStack``
     * a red heart ``Image`` defines its
```

```
own ``TapGesture``
    /// handler that also prints a
message to the console, and blue
rectangle
    /// with no custom gesture handlers.
Tapping or clicking the image
    /// prints a message to the console
from the tap gesture handler on the
    /// image, while tapping or clicking
the rectangle inside the ``VStack``
    /// prints a message in the console
from the enclosing vertical stack
    /// gesture handler.
    ///
    /// You can also use the
``isEnabled`` parameter to conditionally
disable
    /// the gesture.
    ///
    ///     struct GestureExample: View {
    ///         @State private var
message = "Message"
    ///         var isGestureEnabled:
Bool
    ///         let newGesture =
TapGesture().onEnded {
    ///             print("Tap on
VStack.")
    ///         }
    ///
    ///         var body: some View {
    ///             VStack(spacing:25) {
    ///                 Image(systemName:
```



```
    /// Attaches a gesture to the view
    // with a higher precedence than gestures
    /// defined by the view.
    ///
    /// Use this method when you need to
    define a high priority gesture
    /// to take precedence over the
    view's existing gestures. The
    /// example below defines a custom
    gesture that prints a message to the
    /// console and attaches it to the
    view's ``VStack``. Inside the ``VStack``
    /// a red heart ``Image`` defines its
    own ``TapGesture`` handler that
    /// also prints a message to the
    console, and a blue rectangle
    /// with no custom gesture handlers.
    Tapping or clicking any of the
    /// views results in a console
    message from the high priority gesture
    /// attached to the enclosing
    ``VStack``.
    ///
    /// You can also use the
    ``isEnabled`` parameter to conditionally
    disable
    /// the gesture.
    ///
    /// struct
HighPriorityGestureExample: View {
    /// @State private var
message = "Message"
```

```
    /**
     * var isGestureEnabled: Bool
     * let newGesture = TapGesture().onEnded {
     *     print("Tap on VStack.")
     * }
     */
     var body: some View {
     /**
      * VStack(spacing:25) {
      *     Image(systemName: "heart.fill")
      *         .resizable()
      *         .frame(width: 75, height: 75)
      *         .padding()
      *         .foregroundCo
      * lor(.red)
      *         .onTapGesture
      * {
      *     /**
      * print("Tap on image.")
      *         }
      *         Rectangle()
      *             .fill(Color.b
      * lue)
      *         }
      *         .highPriorityGesture(
      *             newGesture,
      * isEnabled: isGestureEnabled)
      *         .frame(width: 200,
      * height: 200)
      *         .border(Color.purple)
      */
```

```
    /**
     * - Parameters:
     *   - gesture: A gesture to attach to the view.
     *   - isEnabled: Whether the added gesture is enabled.
     */
    nonisolated public func
highPriorityGesture<T>(_ gesture: T,
isEnabled: Bool) -> some View where T : Gesture
```

```
    /**
     * Attaches a gesture to the view to process simultaneously with gestures
     * defined by the view.
     */
    /**
     * Use this method when you need to define and process a view specific
     * gesture simultaneously with the same priority as the
     * view's existing gestures. The example below defines a custom gesture
     * that prints a message to the console and attaches it to the view's
     * ``VStack``. Inside the ``VStack`` is a red heart ``Image`` defines its
     * own ``TapGesture`` handler that also prints a message to the console
     * and a blue rectangle with no custom gesture handlers.
     */
```

```
    /// You can also use the
    ``isEnabled`` parameter to conditionally
    disable
        /// the gesture.
        ///
        /// Tapping or clicking the "heart"
    image sends two messages to the
        /// console: one for the image's tap
    gesture handler, and the other from a
        /// custom gesture handler attached
    to the enclosing vertical stack.
        /// Tapping or clicking on the blue
    rectangle results only in the single
        /// message to the console from the
    tap recognizer attached to the
        /// ``VStack``:
        ///
        ///     struct
SimultaneousGestureExample: View {
    ///         @State private var
message = "Message"
    ///         var isGestureEnabled:
Bool
    ///             let newGesture =
TapGesture().onEnded {
    ///                 print("Gesture on
VStack.")
    ///             }
    ///
    ///         var body: some View {
    ///             VStack(spacing:25) {
    ///                 Image(systemName:
"heart.fill")
```



```
}
```

```
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension View {  
  
    /// Attaches a gesture to the view  
    /// with a lower precedence than gestures  
    /// defined by the view.  
    ///  
    /// Use this method when you need to  
    /// attach a gesture to a view. The  
    /// example below defines a custom  
    /// gesture that prints a message to the  
    /// console and attaches it to the  
    /// view's ``VStack``. Inside the ``VStack``  
    /// a red heart ``Image`` defines its  
    /// own ``TapGesture``  
    /// handler that also prints a  
    /// message to the console, and blue  
    /// rectangle  
    /// with no custom gesture handlers.  
    /// Tapping or clicking the image  
    /// prints a message to the console  
    /// from the tap gesture handler on the  
    /// image, while tapping or clicking  
    /// the rectangle inside the ``VStack``  
    /// prints a message in the console  
    /// from the enclosing vertical stack  
    /// gesture handler.  
    ///  
    /// You can also use the
```

```
``isEnabled`` parameter to conditionally
disable
    /// the gesture.
    ///
    /**
     * struct GestureExample: View {
     *     @State private var
message = "Message"
     *         var isGestureEnabled:
Bool
     *             let newGesture =
TapGesture().onEnded {
     *                 print("Tap on
VStack.")
     *             }
     *
     *             var body: some View {
     *                 VStack(spacing:25) {
     *                     Image(systemName:
"heart.fill")
     *                         .resizable()
     *                         .frame(width:
75, height: 75)
     *                         .padding()
     *                         .foregroundCo
lor(.red)
     *                         .onTapGesture
{
     *                 print("Tap on image.")
     *             }
     *             Rectangle()
     *                 .fill(Color.b
lue)
```

```
    /**
     * - Parameters:
     *   - gesture: A gesture to attach
     *     to the view.
     *   - name: A string that
     *     identifies the gesture. In iOS, the name
     *     can be
     *   - used to set up failure
     *     relationships between UIKit gesture
     *     recognizers and this
     *     gesture.
     *   - isEnabled: Whether the added
     *     gesture is enabled. The default value
     *     is `true`.
     */
    nonisolated public func gesture<T>(_
        gesture: T, name: String, isEnabled: Bool
        = true) -> some View where T : Gesture
```

```
    /**
     * Attaches a gesture to the view
     * with a higher precedence than gestures
     * defined by the view.
     */
    /**
     * Use this method when you need to
     * define a high priority gesture
     */
```

```
    /// to take precedence over the
view's existing gestures. The
    /// example below defines a custom
gesture that prints a message to the
    /// console and attaches it to the
view's ``VStack``. Inside the ``VStack``
    /// a red heart ``Image`` defines its
own ``TapGesture`` handler that
    /// also prints a message to the
console, and a blue rectangle
    /// with no custom gesture handlers.
Tapping or clicking any of the
    /// views results in a console
message from the high priority gesture
    /// attached to the enclosing
``VStack``.
    ///
    /// You can also use the
``isEnabled`` parameter to conditionally
disable
    /// the gesture.
    ///
    /// struct
HighPriorityGestureExample: View {
    ///         @State private var
message = "Message"
    ///         var isGestureEnabled:
Bool
    ///             let newGesture =
TapGesture().onEnded {
    ///                 print("Tap on
VStack.")
    /// }
```

```
///
///         var body: some View {
///             VStack(spacing:25) {
///                 Image(systemName:
/// "heart.fill")
///                 .resizable()
///                 .frame(width:
/// 75, height: 75)
///                 .padding()
///                 .foregroundCo
/// lor(.red)
///                 .onTapGesture
{
///             print("Tap on image.")
///             }
///             Rectangle()
///             .fill(Color.b
lue)
///             }
///             .highPriorityGesture(
///                 newGesture,
/// isEnabled: isGestureEnabled)
///                 .frame(width: 200,
height: 200)
///                 .border(Color.purple)
///             }
///         }
///     }
/// - Parameters:
///     - gesture: A gesture to attach
to the view.
///     - name: A string that
```

identifies the gesture. In iOS, the name can be

```
    /// used to set up failure relationships between UIKit gesture
    /// recognizers and this gesture.
    /// - isEnabled: Whether the added gesture is enabled. The default value
    ///   is `true`.
nonisolated public func
highPriorityGesture<T>(_ gesture: T,
name: String, isEnabled: Bool = true) ->
some View where T : Gesture
```

/// Attaches a gesture to the view to process simultaneously with gestures

/// defined by the view.

///

/// Use this method when you need to define and process a view specific

/// gesture simultaneously with the same priority as the

/// view's existing gestures. The example below defines a custom gesture

/// that prints a message to the console and attaches it to the view's

/// ``VStack``. Inside the ``VStack`` is a red heart ``Image`` defines its

/// own ``TapGesture`` handler that also prints a message to the console

/// and a blue rectangle with no custom gesture handlers.

```
///  
/// You can also use the  
``isEnabled`` parameter to conditionally  
disable  
/// the gesture.  
///  
/// Tapping or clicking the "heart"  
image sends two messages to the  
/// console: one for the image's tap  
gesture handler, and the other from a  
/// custom gesture handler attached  
to the enclosing vertical stack.  
/// Tapping or clicking on the blue  
rectangle results only in the single  
/// message to the console from the  
tap recognizer attached to the  
/// ``VStack``:  
///  
///     struct  
SimultaneousGestureExample: View {  
    ///         @State private var  
message = "Message"  
    ///         var isGestureEnabled:  
Bool  
    ///             let newGesture =  
TapGesture().onEnded {  
    ///                 print("Gesture on  
VStack.")  
    ///             }  
    ///  
    ///         var body: some View {  
    ///             VStack(spacing:25) {  
    ///                 Image(systemName:
```



```
    ///      recognizers and this
gesture.
    ///      - isEnabled: Whether the added
gesture is enabled. The default value
    ///      is `true`.
    nonisolated public func
simultaneousGesture<T>(_ gesture: T,
name: String, isEnabled: Bool = true) ->
some View where T : Gesture

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension View {

    /// Scales this view's rendered
output by the given vertical and
horizontal
    /// size amounts, relative to an
anchor point.
    ///
    /// Use `scaleEffect(_:anchor:)` to
scale a view by applying a scaling
    /// transform of a specific size,
specified by `scale`.
    ///
    ///      Image(systemName:
"envelope.badge.fill")
    ///          .resizable()
    ///          .frame(width: 100,
height: 100, alignment: .center)
    ///          .foregroundColor(Color.re
```

```
d)
    ///           .scaleEffect(CGSize(x:
0.9, y: 1.3), anchor: .leading)
    ///           .border(Color.gray)
    ///
    /// ! [A screenshot showing a red
envelope scaled to a size of 90x130
    /// pixels.] (SwiftUI-View-
scaleEffect.png)
    ///
    /// - Parameters:
    ///   - scale: A
<doc://com.apple.documentation/documentation/CoreFoundation/CGSize> that
    ///     represents the horizontal and
vertical amount to scale the view.
    ///     - anchor: The point with a
default of ``UnitPoint/center`` that
    ///     defines the location within
the view from which to apply the
    ///     transformation.
@inlinable nonisolated public func
scaleEffect(_ scale: CGSize, anchor:
UnitPoint = .center) -> some View
```

```
    /// Scales this view's rendered
output by the given amount in both the
    /// horizontal and vertical
directions, relative to an anchor point.
    ///
    /// Use `scaleEffect(_:anchor:)` to
apply a horizontally and vertically
```

```
    /// scaling transform to a view.  
    ///  
    ///     Image(systemName:  
"envelope.badge.fill")  
    ///         .resizable()  
    ///         .frame(width: 100,  
height: 100, alignment: .center)  
    ///         .foregroundColor(Color.re  
d)  
    ///         .scaleEffect(2,  
anchor: .leading)  
    ///         .border(Color.gray)  
    ///  
    ///     ! [A screenshot showing a 100x100  
pixel red envelope scaled up to 2x the  
    /// size of its view.] (SwiftUI-View-  
scaleEffect-cgfloat.png)  
    ///  
    /// - Parameters:  
    ///     - s: The amount to scale the  
view in the view in both the horizontal  
    ///     and vertical directions.  
    ///     - anchor: The anchor point with  
a default of ``UnitPoint/center`` that  
    ///     indicates the starting  
position for the scale operation.  
    @inlinable nonisolated public func  
scaleEffect(_ s: CGFloat, anchor:  
UnitPoint = .center) -> some View
```

```
    /// Scales this view's rendered  
output by the given horizontal and
```

```
vertical
    /// amounts, relative to an anchor
point.
    /**
     /// Use `scaleEffect(x:y:anchor:)` to
apply a scaling transform to a view by
     /// a specific horizontal and
vertical amount.
    /**
     ///      Image(systemName:
"envelope.badge.fill")
    /**
         .resizable()
    /**
         .frame(width: 100,
height: 100, alignment: .center)
    /**
         .foregroundColor(Color.re
d)
    /**
         .scaleEffect(x: 0.5, y:
0.5, anchor: .bottomTrailing)
    /**
         .border(Color.gray)
    /**
     /// ! [A screenshot showing a 100x100
pixel red envelope scaled down 50% in
     /// both the x and y axes.] (SwiftUI-
View-scaleEffect-xy.png)
    /**
     /// - Parameters:
    /**
        - x: An amount that represents
the horizontal amount to scale the
    /**
        view. The default value is
`1.0`.
    /**
        - y: An amount that represents
the vertical amount to scale the view.
    /**
        The default value is `1.0`.
```

```
    /// - anchor: The anchor point that
    indicates the starting position for
    /// the scale operation.
    @inlinable nonisolated public func
scaleEffect(x: CGFloat = 1.0, y: CGFloat
= 1.0, anchor: UnitPoint = .center) ->
some View

}

@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
extension View {

    /// Shows the specified content above
    or below the modified view.
    ///
    /// The `content` view is anchored to
    the specified
    /// vertical edge in the parent view,
    aligning its horizontal axis
    /// to the specified alignment guide.
    The modified view is inset by
    /// the height of `content`, from
    `edge`, with its safe area
    /// increased by the same amount.
    ///
    ///     struct
ScrollViewWithBottomBar: View {
    ///         var body: some View {
    ///             ScrollView {
    ///                 ScrolledContent()
    ///             }
    ///         }
    ///     }
}
```

```
    ///          .safeAreaInset(edge:  
.bottom, spacing: 0) {  
    ///  
BottomBarContent()  
    ///          }  
    ///          }  
    ///          }  
    ///  
    /// - Parameters:  
    ///   - edge: The vertical edge of  
the view to inset by the height of  
    ///   `content`, to make space for  
`content`.  
    ///   - spacing: Extra distance  
placed between the two views, or  
    ///   nil to use the default amount  
of spacing.  
    ///   - alignment: The alignment  
guide used to position `content`  
    ///   horizontally.  
    ///   - content: A view builder  
function providing the view to  
    ///   display in the inset space of  
the modified view.  
    ///  
    /// - Returns: A new view that  
displays both `content` above or below  
the  
    ///   modified view,  
    ///   making space for the `content`  
view by vertically insetting  
    ///   the modified view, adjusting  
the safe area of the result to match.
```

```
@inlinable nonisolated public func
safeAreaInset<V>(edge: VerticalEdge,
alignment: HorizontalAlignment = .center,
spacing: CGFloat? = nil, @ViewBuilder
content: () -> V) -> some View where V : View
```

```
    /// Shows the specified content
    /// beside the modified view.
    ///
    /// The `content` view is anchored to
    /// the specified
    /// horizontal edge in the parent
    /// view, aligning its vertical axis
    /// to the specified alignment guide.
    The modified view is inset by
    /// the width of `content`, from
    /// `edge`, with its safe area
    /// increased by the same amount.
    ///
    ///     struct
ScrollViewWithSideBar: View {
    ///         var body: some View {
    ///             ScrollView {
    ///                 ScrolledContent()
    ///             }
    ///             .safeAreaInset(edge:
    .leading, spacing: 0) {
    ///                 SideBarContent()
    ///             }
    ///         }
    ///     }
    /// }
```

```
    /**
     * - Parameters:
     *   - edge: The horizontal edge of
     *     the view to inset by the width of
     *     `content`, to make space for
     *     `content`.
     *   - spacing: Extra distance
     *     placed between the two views, or
     *     nil to use the default amount
     *     of spacing.
     *   - alignment: The alignment
     *     guide used to position `content`
     *     vertically.
     *   - content: A view builder
     *     function providing the view to
     *     display in the inset space of
     *     the modified view.
     *
     * - Returns: A new view that
     * displays `content` beside the modified
     * view,
     *   making space for the `content`
     * view by horizontally insetting
     *   the modified view.
     */
    @inlinable nonisolated public func
    safeAreaInset<V>(edge: HorizontalEdge,
                      alignment: VerticalAlignment = .center,
                      spacing: CGFloat? = nil, @ViewBuilder
                      content: () -> V) -> some View where V : View
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension View {  
  
    /// Adds the provided insets into the  
    safe area of this view.  
    ///  
    /// Use this modifier when you would  
    like to add a fixed amount  
    /// of space to the safe area a view  
    sees.  
    ///  
    /// ScrollView(.horizontal) {  
    ///     HStack(spacing: 10.0) {  
    ///         ForEach(items) { item  
    in  
        ///             ItemView(item)  
        ///         }  
        ///     }  
        /// }  
        /// .safeAreaPadding(.horizontal,  
20.0)  
    ///  
    /// See the  
``View/safeAreaInset(edge:alignment:spaci  
ng:content)``  
    /// modifier for adding to the safe  
    area based on the size of a  
    /// view.  
    @available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
    nonisolated public func  
safeAreaPadding(_ insets: EdgeInsets) ->
```

## some View

```
    /// Adds the provided insets into the
    /// safe area of this view.
    ///
    /// Use this modifier when you would
    like to add a fixed amount
    /// of space to the safe area a view
sees.
    ///
    /// ScrollView(.horizontal) {
    ///     HStack(spacing: 10.0) {
    ///         ForEach(items) { item
in
        ///
        ///             ItemView(item)
        ///
        ///         }
        ///
        ///     }
        ///
        ///     .safeAreaPadding(.horizontal,
20.0)
        ///
        /// See the
``View/safeAreaInset(edge:alignment:spaci
ng:content)``
        /// modifier for adding to the safe
area based on the size of a
        /// view.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    nonisolated public func
safeAreaPadding(_ edges: Edge.Set = .all,
_ length: CGFloat? = nil) -> some View
```

```
    /// Adds the provided insets into the
    safe area of this view.
    ///
    /// Use this modifier when you would
    like to add a fixed amount
    /// of space to the safe area a view
    sees.
    ///
    /// ScrollView(.horizontal) {
    ///     HStack(spacing: 10.0) {
    ///         ForEach(items) { item
    in
    ///             ItemView(item)
    ///         }
    ///     }
    /// }
    /// .safeAreaPadding(.horizontal,
20.0)
    ///
    /// See the
``View/safeAreaInset(edge:alignment:spaci
ng:content)``
    /// modifier for adding to the safe
    area based on the size of a
    /// view.
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
    nonisolated public func
safeAreaPadding(_ length: CGFloat) ->
some View
```

```
}
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
extension View {  
  
    /// Sets the tint within this view.  
    ///  
    /// Use this method to override the  
    default accent color for this view with  
    /// a given styling. Unlike an app's  
    accent color, which can be  
    /// overridden by user preference,  
    tint is always respected and should  
    /// be used as a way to provide  
    additional meaning to the control.  
    ///  
    /// Controls which are unable to  
    style themselves using the given type of  
    /// `ShapeStyle` will try to  
    approximate the styling as best as they  
    can  
    /// (i.e. controls which can not  
    style themselves using a gradient will  
    /// attempt to use the color of the  
    gradient's first stop).  
    ///  
    /// This example shows a linear gauge  
    tinted with a  
    /// gradient from ``ShapeStyle/blue``  
    to ``ShapeStyle/red``.  
    ///  
    ///     struct ControlTint: View {
```

```
    /**
     * var body: some View {
     *   Gauge(value: 75, in:
     * 0...100) {
     *   /**
     *     Text("Temperature")
     *   /**
     *     /**
     *       .gaugeStyle(.linearCa
     * pacity)
     *     /**
     *       .tint(Gradient(colors
     * : [.blue, .orange, .red]))
     *     /**
     *   }
     * /**
     *   /**
     *     /**
     *       Some controls adapt to the tint
     * color differently based on their style,
     *     /**
     *       the current platform, and the
     * surrounding context. For example, in
     *     /**
     *       macOS, a button with the
     * ``PrimitiveButtonStyle/bordered`` style
     * doesn't
     *     /**
     *       tint its background, but one with
     * the
     *     /**
     * ``PrimitiveButtonStyle/borderedProminent``
     * ` style does. In macOS, neither
     *     /**
     *       of these button styles tint their
     * label, but they do in other platforms.
     * /**
     *     /**
     *       - Parameter tint: The tint to
     * apply.
     * @inlinable nonisolated public func
     * tint<S>(_ tint: S?) -> some View where
     * S : ShapeStyle
```

```
}
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)
```

```
extension View {
```

```
    /// Sets the tint color within this  
view.
```

```
    ///
```

```
    /// Use this method to override the  
default accent color for this view.
```

```
    /// Unlike an app's accent color,  
which can be overridden by user  
    /// preference, the tint color is  
always respected and should be used as a  
    /// way to provide additional meaning  
to the control.
```

```
    ///
```

```
    /// This example shows Answer and  
Decline buttons with ``ShapeStyle/green``  
    /// and ``ShapeStyle/red`` tint  
colors, respectively.
```

```
    ///
```

```
    /// struct ControlTint: View {
```

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

```
    ///         HStack {
```

```
    ///             Button {
```

```
    ///                 // Answer the  
call
```

```
    ///             } label: {
```

```
    ///             Label("Answer", systemImage: "phone")
```





```
colors: [.blue, .red, .green],  
        ////  
startPoint: .top, endPoint: .bottom))  
        ////  
        .hueRotation(.degrees(Double($0 * 36))))  
        ////  
        .frame(width: 60, height: 60, alignment: .center)  
        ////  
        }  
        ////  
        }  
        ////  
        }  
        ////  
        /// !-[Shows the effect of hueRotation  
on a linear  
        /// gradient.] (SwiftUI-  
hueRotation.png)  
        ////  
        /// - Parameter angle: The hue  
rotation angle to apply to the colors in  
this  
        /// view.  
        ////  
        /// - Returns: A view that applies a  
hue rotation effect to this view.  
    @inlinable nonisolated public func  
hueRotation(_ angle: Angle) -> some View  
  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension View {
```

```
    /// Associates a transition with the
view.
    /**
     * When this view appears or
disappears, the transition will be
applied to
     * it, allowing for animating it in
and out.
    /**
     * The following code will
conditionally show MyView, and when it
appears
     * or disappears, will use a slide
transition to show it.
    /**
     /**
     if isActive {
     /**
         MyView()
     /**
         .transition(.slide)
     /**
     }
     /**
     Button("Toggle") {
     /**
         withAnimation {
     /**
             isActive.toggle()
     /**
         }
     /**
     }
@inlinable nonisolated public func
transition(_ t: AnyTransition) -> some
View
```

```
    /// Associates a transition with the
view.
    /**
     * When this view appears or
```

```
disappears, the transition will be
applied to
    /// it, allowing for animating it in
and out.
    ///
    /// The following code will
conditionally show MyView, and when it
appears
    /// or disappears, will use a custom
RotatingFadeTransition transition to
    /// show it.
    ///
    ///     if isActive {
    ///         MyView()
    ///             .transition(RotatingF
adeTransition())
    ///     }
    ///     Button("Toggle") {
    ///         withAnimation {
    ///             isActive.toggle()
    ///         }
    ///     }
    @available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
nonisolated public func
transition<T>(_ transition: T) -> some
View where T : Transition

}

/// A custom parameter attribute that
constructs views from closures.
///
```

```
/// You typically use ``ViewBuilder`` as
/// a parameter attribute for child
/// view-producing closure parameters,
/// allowing those closures to provide
/// multiple child views. For example,
/// the following `contextMenu` function
/// accepts a closure that produces one
/// or more views via the view builder.
///
///     func contextMenu<MenuItem: View>(
///         @ViewBuilder menuItems: () -> MenuItem
///     ) -> some View
///
/// Clients of this function can use
/// multiple-statement closures to provide
/// several child views, as shown in the
/// following example:
///
///     myView.contextMenu {
///         Text("Cut")
///         Text("Copy")
///         Text("Paste")
///         if isSymbol {
///             Text("Jump to
/// Definition")
///         }
///     }
///
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@resultBuilder public struct ViewBuilder
```

```
{  
  
    /// Builds an expression within the  
    builder.  
    public static func  
buildExpression<Content>(_ content:  
Content) -> Content where Content : View  
  
    /// Builds an empty view from a block  
containing no statements.  
    public static func buildBlock() ->  
EmptyView  
  
    /// Passes a single view written as a  
child view through unmodified.  
    ///  
    /// An example of a single view  
written as a child view is  
    /// ` { Text("Hello") }`.  
    public static func  
buildBlock<Content>(_ content: Content)  
-> Content where Content : View  
  
    public static func buildBlock<each  
Content>(_ content: repeat each Content)  
-> TupleView<(repeat each Content)> where  
repeat each Content : View  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ViewBuilder {
```

```
    /// Produces an optional view for
conditional statements in multi-statement
    /// closures that's only visible when
the condition evaluates to true.
    public static func buildIf<Content>(_
content: Content?) -> Content? where
Content : View

    /// Produces content for a
conditional statement in a multi-
statement closure
    /// when the condition is true.
    public static func
buildEither<TrueContent,
FalseContent>(first: TrueContent) ->
    _ConditionalContent<TrueContent,
FalseContent> where TrueContent : View,
FalseContent : View

    /// Produces content for a
conditional statement in a multi-
statement closure
    /// when the condition is false.
    public static func
buildEither<TrueContent,
FalseContent>(second: FalseContent) ->
    _ConditionalContent<TrueContent,
FalseContent> where TrueContent : View,
FalseContent : View
}

@available(iOS 14.0, macOS 11.0, tvOS
14.0, watchOS 7.0, *)
```

```
extension ViewBuilder {  
  
    /// Processes view content for a  
    conditional compiler-control  
    /// statement that performs an  
    availability check.  
    public static func  
buildLimitedAvailability<Content>(_  
content: Content) -> AnyView where  
Content : View  
}  
  
/// A view's size and alignment guides in  
its own coordinate space.  
///  
/// This structure contains the size and  
alignment guides of a view.  
/// You receive an instance of this  
structure to use in a variety of  
/// layout calculations, like when you:  
///  
/// * Define a default value for a custom  
alignment guide;  
/// see  
``AlignmentID/defaultValue(in:)``.  
/// * Modify an alignment guide on a  
view;  
/// see  
``View/alignmentGuide(_:computeValue:)``.  
/// * Ask for the dimensions of a subview  
of a custom view layout;  
/// see  
``LayoutSubview/dimensions(in:)``.
```

```
///  
/// Custom alignment guides  
///  
/// You receive an instance of this  
structure as the `context` parameter to  
/// the ``AlignmentID/defaultValue(in:)``  
method that you implement to produce  
/// the default offset for an alignment  
guide, or as the first argument to the  
/// closure you provide to the  
``View/alignmentGuide(_:computeValue:)``  
/// view modifier to override the default  
calculation for an alignment guide.  
/// In both cases you can use the  
instance, if helpful, to calculate the  
/// offset for the guide. For example,  
you could compute a default offset  
/// for a custom ``VerticalAlignment`` as  
a fraction of the view's ``height``:  
///  
///     private struct  
FirstThirdAlignment: AlignmentID {  
///         static func defaultValue(in  
context: ViewDimensions) -> CGFloat {  
///             context.height / 3  
///         }  
///     }  
///  
///     extension VerticalAlignment {  
///         static let firstThird =  
VerticalAlignment(FirstThirdAlignment.self)  
///     }
```

```
///  
/// As another example, you could use the  
view dimensions instance to look  
/// up the offset of an existing guide  
and modify it:  
///  
///     struct ViewDimensionsOffset: View  
{  
///         var body: some View {  
///             VStack(alignment: .leading) {  
///                 Text("Default")  
///                 Text("Indented")  
///                     .alignmentGuide(.  
leading) { context in  
///                     context[.leading] - 10  
///                         }  
///                     }  
///             }  
///  
/// The example above indents the second  
text view because the subtraction  
/// moves the second text view's leading  
guide in the negative x direction,  
/// which is to the left in the view's  
coordinate space. As a result,  
/// SwiftUI moves the second text view to  
the right, relative to the first  
/// text view, to keep their leading  
guides aligned:  
///
```

```
/// ! [A screenshot of two strings. The
/// first says Default and the second,
/// which appears below the first, says
/// Indented. The left side of the second
/// string appears horizontally offset to
/// the right from the left side of the
/// first string by about the width of
/// one character.] (ViewDimensions-1-iOS)
///
/// Layout direction
///
/// The discussion above describes a
/// left-to-right language environment,
/// but you don't change your guide
/// calculation to operate in a right-to-left
/// environment. SwiftUI moves the view's
/// origin from the left to the right side
/// of the view and inverts the positive
/// x direction. As a result,
/// the existing calculation produces the
/// same effect, but in the opposite
/// direction.
///
/// You can see this if you use the
``View/environment(_:_)``
/// modifier to set the
``EnvironmentValues/layoutDirection``
property for the
/// view that you defined above:
///
///     ViewDimensionsOffset()
///         .environment(\.layoutDirection,
///         .rightToLeft)
```

```
///  
/// With no change in your guide, this  
produces the desired effect ---  
/// it indents the second text view's  
right side, relative to the  
/// first text view's right side. The  
leading edge is now on the right,  
/// and the direction of the offset is  
reversed:  
///  
/// ! [A screenshot of two strings. The  
first says Default and the second,  
/// which appears below the first, says  
Indented. The right side of the second  
/// string appears horizontally offset to  
the left from the right side of the  
/// first string by about the width of  
one character.] (ViewDimensions-2-iOS)  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
public struct ViewDimensions {  
  
    /// The view's width.  
    public var width: CGFloat { get }  
  
    /// The view's height.  
    public var height: CGFloat { get }  
  
    /// Gets the value of the given  
horizontal guide.  
    ///  
    /// Find the offset of a particular  
guide in the corresponding view by
```

```
    /// using that guide as an index to
    // read from the context:
    /**
     ///      .alignmentGuide(.leading)
{ context in
    /**
     ///          context[.leading] - 10
    /**
     */
    /**
     /// For information about using
     subscripts in Swift to access member
     /// elements of a collection, list,
     or, sequence, see
     /// [Subscripts]
(https://docs.swift.org/swift-book/LanguageGuide/Subscripts.html)
     /// in The Swift Programming
     Language.
     public subscript(guide:
HorizontalAlignment) -> CGFloat { get }

     /// Gets the value of the given
     vertical guide.
     /**
     /// Find the offset of a particular
     guide in the corresponding view by
     /// using that guide as an index to
     read from the context:
     /**
     ///      .alignmentGuide(.top)
{ context in
    /**
     ///          context[.top] - 10
    /**
     */
    /**

```

```
    /// For information about using
    subscripts in Swift to access member
    /// elements of a collection, list,
or, sequence, see
    /// [Subscripts]
(https://docs.swift.org/swift-book/LanguageGuide/Subscripts.html)
    /// in The Swift Programming Language.
public subscript(guide:
VerticalAlignment) -> CGFloat { get }

    /// Gets the explicit value of the
given horizontal alignment guide.
    ///
    /// Find the horizontal offset of a
particular guide in the corresponding
    /// view by using that guide as an
index to read from the context:
    ///
    ///     .alignmentGuide(.leading)
{ context in
    ///         context[.leading] - 10
    ///
    ///
    /// This subscript returns `nil` if
no value exists for the guide.
    ///
    /// For information about using
subscripts in Swift to access member
    /// elements of a collection, list,
or, sequence, see
    /// [Subscripts]
```

(<https://docs.swift.org/swift-book/LanguageGuide/Subscripts.html>)  
    /// in *The Swift Programming Language*.  
    **public subscript**(**explicit guide:**  
        HorizontalAlignment) -> CGFloat? { **get** }  
  
        /// Gets the explicit value of the  
        given vertical alignment guide  
        ///  
        /// Find the vertical offset of a  
        particular guide in the corresponding  
        /// view by using that guide as an  
        index to read from the context:  
        ///  
        ///     .alignmentGuide(.top)  
    { context in  
        ///     context[.top] - 10  
        ///     }  
        ///  
        /// This subscript returns `nil` if  
        no value exists for the guide.  
        ///  
        /// For information about using  
        subscripts in Swift to access member  
        /// elements of a collection, list,  
        or, sequence, see  
        /// [Subscripts]  
(<https://docs.swift.org/swift-book/LanguageGuide/Subscripts.html>)  
    /// in *The Swift Programming Language*.  
    **public subscript**(**explicit guide:**

```
VerticalAlignment) -> CGFloat? { get }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ViewDimensions : Equatable {

    /// 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 == (lhs:
    ViewDimensions, rhs: ViewDimensions) ->
    Bool
}

/// A modifier that you apply to a view
or another view modifier, producing a
/// different version of the original
value.
///
/// Adopt the ``ViewModifier`` protocol
when you want to create a reusable
/// modifier that you can apply to any
view. The example below combines several
```

```
/// modifiers to create a new modifier
/// that you can use to create blue caption
/// text surrounded by a rounded
rectangle:
///
///     struct BorderedCaption:
ViewModifier {
    func body(content: Content)
-> some View {
    content
        .font(.caption2)
        .padding(10)
        .overlay(
        )
RoundedRectangle(cornerRadius: 15)
        .stroke(lineWidth: 1)
        .foregroundColor(Color
r.blue)
    }
}
///
/// You can apply ``View/modifier(_:)``
directly to a view, but a more common
/// and idiomatic approach uses
``View/modifier(_:)`` to define an
extension to
/// ``View`` itself that incorporates the
view modifier:
///
///     extension View {
///         func borderedCaption() ->
```

```
some View {
    /**
     modifier(BorderedCaption())
     */
}
/// You can then apply the bordered
/// caption to any view, similar to this:
///
///     Image(systemName: "bus")
///         .resizable()
///         .frame(width:50, height:50)
///     Text("Downtown Bus")
///         .borderedCaption()
///
/// ! [A screenshot showing the image of a
/// bus with a caption reading
/// Downtown Bus. A view extension, using
/// custom a modifier, renders the
/// caption in blue text surrounded by a
/// rounded
/// rectangle.] (SwiftUI-View-
/// ViewModifier.png)
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@MainActor @preconcurrency public
protocol ViewModifier {

    /// The type of view representing the
    /// body.
    associatedtype Body : View

    /// Gets the current body of the
```

```
caller.  
    ///  
    /// `content` is a proxy for the view  
    /// that will have the modifier  
    /// represented by `Self` applied to  
    /// it.  
    @ViewBuilder @MainActor  
    @preconcurrency func body(content:  
        Self.Content) -> Self.Body  
  
        /// The content view type passed to  
        /// `body()`.  
        typealias Content  
    }  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension ViewModifier where Self.Body ==  
Never {  
  
    /// Gets the current body of the  
    /// caller.  
    ///  
    /// `content` is a proxy for the view  
    /// that will have the modifier  
    /// represented by `Self` applied to  
    /// it.  
    @MainActor @preconcurrency public  
    func body(content: Self.Content) ->  
        Self.Body  
    }  
  
@available(iOS 13.0, macOS 10.15, tvOS
```

```
13.0, watchOS 6.0, *)
extension ViewModifier {

    /// Returns a new modifier that is
    the result of concatenating
    /// `self` with `modifier`.
    @inlinable nonisolated public func
concat<T>(_ modifier: T) ->
ModifiedContent<Self, T>
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension ViewModifier {

    /// Returns a new version of the
    modifier that will apply the
    /// transaction mutation function
    `transform` to all transactions
    /// within the modifier.
    @inlinable nonisolated public func
transaction(_ transform: @escaping (inout
Transaction) -> Void) -> some
ViewModifier

    /// Returns a new version of the
    modifier that will apply
    /// `animation` to all animatable
    values within the modifier.
    @MainActor @inlinable @preconcurrency
public func animation(_ animation:
Animation?) -> some ViewModifier
```

```
}

/// A collection of the geometric spacing
preferences of a view.
///
/// This type represents how much space a
view prefers to have between it and
/// the next view in a layout. The type
stores independent values
/// for each of the top, bottom, leading,
and trailing edges,
/// and can also record different values
for different kinds of adjacent
/// views. For example, it might contain
one value for the spacing to the next
/// text view along the top and bottom
edges, other values for the spacing to
/// text views on other edges, and yet
other values for other kinds of views.
/// Spacing preferences can also vary by
platform.
///
/// Your ``Layout`` type doesn't have to
take preferred spacing into
/// account, but if it does, you can use
the ``LayoutSubview/spacing``
/// preferences of the subviews in your
layout container to:
///
/// * Add space between subviews when you
implement the
///
```

```
``Layout/placeSubviews(in:proposal:subviews:cache:)`` method.  
/// * Create a spacing preferences instance for the container view by  
///   implementing the  
``Layout/spacing(subviews:cache:)``  
method.  
@available(iOS 16.0, macOS 13.0, tvOS 16.0, watchOS 9.0, *)  
public struct ViewSpacing : Sendable {  
  
    /// A view spacing instance that contains zero on all edges.  
    ///  
    /// You typically only use this value for an empty view.  
    public static let zero: ViewSpacing  
  
    /// Initializes an instance with default spacing values.  
    ///  
    /// Use this initializer to create a spacing preferences instance with  
    /// default values. Then use  
    ``formUnion(_:edges:)`` to combine  
    /// preferences from other views with the new instance. You typically  
    /// do this in a custom layout's implementation of the  
    ///  
    ``Layout/spacing(subviews:cache:)``  
method.  
public init()
```

```
    /// Merges the spacing preferences of
    /// another spacing instance with this
    /// instance for a specified set of
edges.
    ///
    /// When you merge another spacing
    preference instance with this one,
    /// this instance ends up with the
    greater of its original value or the
    /// other instance's value for each
    of the specified edges.
    /// You can call the method
    repeatedly with each value in a
    collection to
    /// merge a collection of
    preferences. The result has the smallest
    /// preferences on each edge that
    meets the largest requirements of all
    /// the inputs for that edge.
    ///
    /// If you want to merge preferences
    without modifying the original
    /// instance, use ``union(_:edges:)`` instead.
    ///
    /// - Parameters:
    ///   - other: Another spacing
    preferences instances to merge with this
    one.
    ///   - edges: The edges to merge.
    Edges that you don't specify are
    ///   unchanged after the method
```

completes.

```
public mutating func formUnion(_  
other: ViewSpacing, edges: Edge.Set  
= .all)
```

    /// Gets a new value that merges the  
    spacing preferences of another spacing

    /// instance with this instance for a  
    specified set of edges.

    ///

    /// This method behaves like  
``formUnion(\_:edges:)``, except that it  
creates

    /// a copy of the original spacing  
    preferences instance before merging,

    /// leaving the original instance  
    unmodified.

    ///

    /// - Parameters:

    /// - other: Another spacing  
    preferences instance to merge with this  
    one.

    /// - edges: The edges to merge.  
    Edges that you don't specify are

    /// unchanged after the method  
    completes.

    ///

    /// - Returns: A new view spacing  
    preferences instance with the merged

    /// values.

```
public func union(_ other:  
ViewSpacing, edges: Edge.Set = .all) ->  
ViewSpacing
```

```
    /// Gets the preferred spacing
    distance along the specified axis to the
    view
        /// that returns a specified spacing
        preference.
        ///
        /// Call this method from your
        implementation of ``Layout`` protocol
        /// methods if you need to measure
        the default spacing between two
        /// views in a custom layout. Call
        the method on the first view's
        /// preferences instance, and provide
        the second view's preferences
        /// instance as input.
        ///
        /// For example, consider two views
        that appear in a custom horizontal
        /// stack. The following distance
        call gets the preferred spacing between
        /// these views, where `spacing1`
        contains the preferences of a first
        /// view, and `spacing2` contains the
        preferences of a second view:
        ///
        ///     let distance =
        spacing1.distance(to: spacing2,
        axis: .horizontal)
        ///
        /// The method first determines,
        based on the axis and the ordering, that
        /// the views abut on the trailing
```

```
edge of the first view and the leading
    /// edge of the second. It then gets
the spacing preferences for the
    /// corresponding edges of each view,
and returns the greater of the two
    /// values. This results in the
smallest value that provides enough space
    /// to satisfy the preferences of
both views.

    ///
    /// > Note: This method returns the
default spacing between views, but a
    /// layout can choose to ignore the
value and use custom spacing instead.

    ///
    /// - Parameters:
    ///   - next: The spacing preferences
instance of the adjacent view.
    ///   - axis: The axis that the two
views align on.

    ///
    /// - Returns: A floating point value
that represents the smallest distance
    /// in points between two views
that satisfies the spacing preferences
    /// of both this view and the
adjacent views on their shared edge.

public func distance(to next:
ViewSpacing, along axis: Axis) -> CGFloat
}

/// The visibility of a UI element,
chosen automatically based on
```

```
/// the platform, current context, and
other factors.
///
/// For example, the preferred visibility
of list row separators can be
/// configured using the
``View/listRowSeparator(_:edges:)``.
@available(iOS 15.0, macOS 12.0, tvOS
15.0, watchOS 8.0, *)
@frozen public enum Visibility :  
Hashable, CaseIterable {
    /// The element may be visible or
hidden depending on the policies of the
    /// component accepting the
visibility configuration.
    ///
    /// For example, some components
employ different automatic behavior
    /// depending on factors including
the platform, the surrounding container,
    /// user settings, etc.
    case automatic

    /// The element may be visible.
    ///
    /// Some APIs may use this value to
represent a hint or preference, rather
    /// than a mandatory assertion. For
example, setting list row separator
    /// visibility to `visible` using the
    ///
``View/listRowSeparator(_:edges:)``
```

```
modifier may not always
    /// result in any visible separators,
especially for list styles that do not
    /// include separators as part of
their design.

case visible

    /// The element may be hidden.
    ///
    /// Some APIs may use this value to
represent a hint or preference, rather
    /// than a mandatory assertion. For
example, setting confirmation dialog
    /// title visibility to `hidden`
using the
    ///
``View/confirmationDialog(_:isPresented:titleVisibility:actions:)``
    /// modifier may not always hide the
dialog title, which is required on
    /// some platforms.

case hidden

    /// 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: Visibility,
b: Visibility) -> Bool

    /// 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)

    /// A type that can represent a
```

```
collection of all values of this type.  
    @available(iOS 15.0, tvOS 15.0,  
watchOS 8.0, macOS 12.0, *)  
    public typealias AllCases =  
[Visibility]  
  
        /// A collection of all values of  
this type.  
        nonisolated public static var  
allCases: [Visibility] { get }  
  
        /// 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 }  
}  
  
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Visibility : Sendable {  
}
```

```
@available(iOS 15.0, macOS 12.0, tvOS  
15.0, watchOS 8.0, *)  
extension Visibility : BitwiseCopyable {  
}  
  
/// Visual Effects change the visual  
appearance of a view without changing its  
/// ancestors or descendants.  
///  
/// Because effects do not impact layout,  
they are safe to use in situations  
/// where layout modification is not  
allowed. For example, effects may be  
/// applied as a function of position,  
accessed through a geometry proxy:  
///  
/// ````swift  
/// var body: some View {  
///     ContentRow()  
///         .visualEffect { content,  
geometryProxy in  
///             content.offset(x:  
geometryProxy.frame(in: .global).origin.y  
}  
///     }  
/// ````  
///  
/// You don't conform to this protocol  
yourself. Instead, visual effects are  
/// created by calling modifier functions  
(such as `offset(x:y:)` on other
```

```
/// effects, as seen in the example  
above.  
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
public protocol VisualEffect : Sendable,  
Animatable {  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension VisualEffect {  
  
    /// Sets the blend mode for  
    // composing this view with overlapping  
    // views.  
    ///  
    /// Use `blendMode(_:)` to combine  
    // overlapping views and use a different  
    /// visual effect to produce the  
    // result. The ``BlendMode`` enumeration  
    /// defines many possible effects.  
    ///  
    /// - Parameter blendMode: The  
    // ``BlendMode`` for compositing.  
    ///  
    /// - Returns: An effect that applies  
    // `blendMode` to this view.  
    public func blendMode(_ blendMode:  
BlendMode) -> some VisualEffect  
  
}  
  
/// A view that overlays its subviews,
```

```
aligning them in both axes.  
///  
/// The `ZStack` assigns each successive  
subview a higher z-axis value than  
/// the one before it, meaning later  
subviews appear "on top" of earlier ones.  
///  
/// The following example creates a  
`ZStack` of 100 x 100 point ``Rectangle``  
/// views filled with one of six colors,  
offsetting each successive subview  
/// by 10 points so they don't completely  
overlap:  
///  
///     let colors: [Color] =  
///  
[.red, .orange, .yellow, .green, .blue, .  
purple]  
///  
///     var body: some View {  
///         ZStack {  
///             ForEach(0..<colors.count)  
{  
///                 Rectangle()  
///                     .fill(colors[$0])  
///                     .frame(width:  
100, height: 100)  
///                     .offset(x:  
CGFloat($0) * 10.0,  
///                             y:  
CGFloat($0) * 10.0)  
///             }  
///         }  
///     }
```

```
///      }
///
/// ! [Six squares of different colors,
/// stacked atop each other, with a 10-point
/// offset in both the x and y axes for
/// each layer so they can be
/// seen.] (SwiftUI-ZStack-offset-
rectangles.png)
///
/// The `ZStack` uses an ``Alignment`` to
/// set the x- and y-axis coordinates of
/// each subview, defaulting to a
/// ``Alignment/center`` alignment. In the
/// following
/// example, the `ZStack` uses a
/// ``Alignment/bottomLeading`` alignment to
/// lay
/// out two subviews, a red 100 x 50
/// point rectangle below, and a blue 50 x
/// 100
/// point rectangle on top. Because of
/// the alignment value, both rectangles
/// share a bottom-left corner with the
/// `ZStack` (in locales where left is the
/// leading side).
///
///     var body: some View {
///
ZStack(alignment: .bottomLeading) {
///             Rectangle()
///                 .fill(Color.red)
///                 .frame(width: 100,
height: 50)
```

```
///             Rectangle()
///             .fill(Color.blue)
///             .frame(width:50,
height: 100)
///         }
///         .border(Color.green, width:
1)
///     }
///
/// ! [A green 100 by 100 square
containing two overlapping rectangles: on
the
/// bottom, a red 100 by 50 rectangle,
and atop it, a blue 50 by 100 rectangle.
/// The rectangles share their bottom
left point with the containing green
/// square.] (SwiftUI-ZStack-
alignment.png)
///
/// > Note: If you need a version of this
stack that conforms to the ``Layout``
/// protocol, like when you want to
create a conditional layout using
/// ``AnyLayout``, use ``ZStackLayout``
instead.
@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
@frozen public struct ZStack<Content> :
View where Content : View {
    /// Creates an instance with the
given alignment.
    ///
```

```
    /// - Parameters:  
    /// - alignment: The guide for  
    aligning the subviews in this stack on  
    both  
    ///      the x- and y-axes.  
    /// - content: A view builder that  
    creates the content of this stack.  
    @inlinable public init(alignment:  
    Alignment = .center, @ViewBuilder  
    content: () -> Content)  
  
    /// The type of view representing the  
    body of this view.  
    ///  
    /// When you create a custom view,  
    Swift infers this type from your  
    /// implementation of the required  
    ``View/body-swift.property`` property.  
    @available(iOS 13.0, tvOS 13.0,  
    watchOS 6.0, macOS 10.15, *)  
    public typealias Body = Never  
}  
  
/// An overlaying container that you can  
use in conditional layouts.  
///  
/// This layout container behaves like a  
``ZStack``, but conforms to the  
/// ``Layout`` protocol so you can use it  
in the conditional layouts that you  
/// construct with ``AnyLayout``. If you  
don't need a conditional layout, use  
/// ``ZStack`` instead.
```

```
@available(iOS 16.0, macOS 13.0, tvOS  
16.0, watchOS 9.0, *)  
@frozen public struct ZStackLayout :  
Layout {  
  
    /// The alignment of subviews.  
    public var alignment: Alignment  
  
        /// Creates a stack with the  
        specified alignment.  
        ///  
        /// - Parameters:  
        ///     - alignment: The guide for  
        aligning the subviews in this stack  
        ///         on both the x- and y-axes.  
        @inlinable public init(alignment:  
Alignment = .center)  
  
            /// The type defining the data to  
            animate.  
            @available(iOS 16.0, tvOS 16.0,  
watchOS 9.0, macOS 13.0, *)  
            public typealias AnimatableData =  
EmptyAnimatableData  
  
            /// Cached values associated with the  
            layout instance.  
            ///  
            /// If you create a cache for your  
            custom layout, you can use  
            /// a type alias to define this type  
            as your data storage type.  
            /// Alternatively, you can refer to
```

```
the data storage type directly in all
    /// the places where you work with
the cache.

    ///
    /// See ``makeCache(subviews:)`` for
more information.

    @available(iOS 16.0, tvOS 16.0,
watchOS 9.0, macOS 13.0, *)
    public typealias Cache = Void
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension ZStackLayout : Sendable {
}

@available(iOS 16.0, macOS 13.0, tvOS
16.0, watchOS 9.0, *)
extension ZStackLayout : BitwiseCopyable
{
}

/// Returns the result of recomputing the
view's body with the provided
/// animation.

///
/// This function sets the given
``Animation`` as the
``Transaction/animation``
/// property of the thread's current
``Transaction``.

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
```

```
public func withAnimation<Result>(_  
    animation: Animation? = .default, _ body:  
    () throws -> Result) rethrows -> Result
```

```
/// Executes a closure with the specified  
transaction and returns the result.
```

```
///
```

```
/// - Parameters:
```

```
///   - transaction : An instance of a  
transaction, set as the thread's current  
///   transaction.
```

```
///   - body: A closure to execute.
```

```
///
```

```
/// - Returns: The result of executing  
the closure with the specified  
///   transaction.
```

```
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)
```

```
public func withTransaction<Result>(_  
    transaction: Transaction, _ body: ()  
throws -> Result) rethrows -> Result
```

```
/// Executes a closure with the specified  
transaction key path and value and  
/// returns the result.
```

```
///
```

```
/// - Parameters:
```

```
///   - keyPath: A key path that  
indicates the property of the  
``Transaction``
```

```
///   structure to update.
```

```
///   - value: The new value to set for  
the item specified by `keyPath`.
```

```
/// - body: A closure to execute.  
///  
/// - Returns: The result of executing  
the closure with the specified  
/// transaction value.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
public func withTransaction<R, V>(_  
keyPath: WritableKeyPath<Transaction, V>,  
_ value: V, _ body: () throws -> R)  
rethrows -> R  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension CGPoint : Animatable {  
  
    /// The type defining the data to  
animate.  
    public typealias AnimatableData =  
AnimatablePair<CGFloat, CGFloat>  
  
    /// The data to animate.  
    public var animatableData:  
CGPoint.AnimatableData  
}  
  
@available(iOS 18.0, macOS 15.0, tvOS  
18.0, watchOS 11.0, visionOS 2.0, *)  
extension FormatStyle where Self ==  
SystemFormatStyle.DateReference {  
  
    /// Create a format style to refer to  
a date in the most natural way.
```

```
///  
/// - Parameters:  
///   - date: The date this format  
references.  
///   - allowedFields: The units of  
time that may be used in the format  
///   to express the reference. The  
`thresholdField` is always assumed  
///   to be allowed.  
///   - maxFieldCount: The number of  
fields that can be shown at once  
///   when using an absolute format.  
For example, January 9, 2007 is  
///   shown as `January 2007` by  
default, but as `January 9, 2007` if  
///   the `maxFieldCount` is set to  
three. The style automatically  
///   excludes more significant  
fields if their value is equal to the  
///   value in the reference date and  
they are not necessary for the  
///   format pattern, making room for  
less significant fields.  
///   - thresholdField: The least  
precise field preserving which  
///   warrants increasing the field  
count from one, i.e. switching from  
///   the relative to the absolute  
representation.  
public static func reference(to date:  
Date, allowedFields:  
Set<Date.RelativeFormatStyle.Field> =  
[.year, .month, .day, .hour, .minute],
```

```
maxFieldCount: Int = 2, thresholdField: Date.RelativeFormatStyle.Field = .day) ->
SystemFormatStyle.DateReference
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension CGSize : Animatable {

    /// The type defining the data to
    animate.
    public typealias AnimatableData =
AnimatablePair<CGFloat, CGFloat>

    /// The data to animate.
    public var animatableData:
CGSize.AnimatableData
}

@available(iOS 17.0, macOS 14.0, tvOS
17.0, watchOS 10.0, *)
extension Never : Keyframes {
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension FormatStyle where Self ==
SystemFormatStyle.DateOffset {

    /// Create a format style to display
    the offset to a certain date.
    ///
    /// The time format (`3:46`) is used
```

as long as only minutes and seconds,  
    /// or hours, minutes, and second may  
be shown.

    /// Otherwise, calendar units are  
used, resulting in outputs like

    /// `3 months, 11 days`.

    ///

    /// The offset to the `anchor` is  
"positive" if the formatted

    /// `date` is greater than the given  
`anchor` specified here.

    /// Conversely, "negative" offsets  
are displayed if the input

    /// `date` is smaller than the  
`anchor`.

    ///

    /// - Parameters:

        /// - anchor: The date the offset  
is measured to from the format input.

        /// - allowedFields: The units of  
time that may be used in the format

        /// to express the offset.

        /// - maxFieldCount: The number of  
fields that can be shown at once.

        /// For example, 1 hour, 34  
minutes, and 23 seconds is shown as

        /// `1 hour, 34 minutes` by  
default, but as `1 hour` if the

        /// `maxFieldCount` is set to one.

        /// - sign: The strategy for  
displaying a sign to signal whether the  
offset

        /// points to ward the future or

```
past.  
    public static func offset(to anchor:  
Date, allowedFields:  
Set<Date.ComponentsFormatStyle.Field> =  
[.year, .month, .day, .hour, .minute, .se  
cond], maxFieldCount: Int = 2, sign:  
NumberFormatStyleConfiguration.SignDispla  
yStrategy = .automatic) ->  
SystemFormatStyle.DateOffset  
}  
  
@available(macOS 12.0, iOS 15.0, tvOS  
15.0, watchOS 8.0, *)  
extension AttributeScopes {  
  
    /// A property for accessing the  
attribute scopes defined by SwiftUI.  
    @available(macOS 12.0, iOS 15.0, tvOS  
15.0, watchOS 8.0, *)  
    public var swiftUI:  
AttributeScopes.SwiftUIAttributes.Type {  
get }  
  
    /// Attribute scopes defined by  
SwiftUI.  
    @available(macOS 12.0, iOS 15.0, tvOS  
15.0, watchOS 8.0, *)  
    public struct SwiftUIAttributes :  
AttributeScope {  
  
        /// A property for accessing a  
font attribute.  
        public let font:
```

## AttributeScopesSwiftUIAttributesFontAttribute

```
    /// A property for accessing a
    foreground color attribute.
    public let foregroundColor:
AttributeScopesSwiftUIAttributesForegroundColorAttribute

    /// A property for accessing a
    background color attribute.
    public let backgroundColor:
AttributeScopesSwiftUIAttributesBackgroundColorAttribute

    /// A property for accessing a
    strikethrough style attribute.
    public let strikethroughStyle:
AttributeScopesSwiftUIAttributesStrikethroughStyleAttribute

    /// A property for accessing an
    underline style attribute.
    public let underlineStyle:
AttributeScopesSwiftUIAttributesUnderlineStyleAttribute

    /// A property for accessing a
    kerning attribute.
    public let kern:
AttributeScopesSwiftUIAttributesKerningAttribute
```

```
    /// A property for accessing a
    tracking attribute.
    public let tracking:
AttributeScopesSwiftUIAttributesTrackingAttribute

    /// A property for accessing a
    baseline offset attribute.
    public let baselineOffset:
AttributeScopesSwiftUIAttributesBaselineOffsetAttribute

    @available(macOS 15.0, iOS 18.0,
tvOS 18.0, watchOS 11.0, visionOS 2.0, *)
    public let adaptiveImageGlyph:
AttributeScopesSwiftUIAttributesAdaptiveImageGlyphAttribute

    /// A property for accessing
    attributes defined by the Accessibility
    framework.
    public let accessibility:
AttributeScopesAccessibilityAttributes

    /// A property for accessing
    attributes defined by the Foundation
    framework.
    public let foundation:
AttributeScopesFoundationAttributes

    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias
```

```
DecodingConfiguration =
AttributeScope Codable Configuration

    @available(iOS 15.0, tvOS 15.0,
watchOS 8.0, macOS 12.0, *)
    public typealias
EncodingConfiguration =
AttributeScope Codable Configuration
}

}

@available(macOS 12.0, iOS 15.0, tvOS
15.0, watchOS 8.0, *)
extension AttributeDynamicLookup {

    public subscript<T>(dynamicMember
keyPath:
KeyPath<AttributeScopes.SwiftUIAttributes
, T>) -> T where T : AttributedStringKey
{ get }
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Float : VectorArithmetic {

    /// Multiplies each component of this
value by the given value.
    public mutating func scale(by rhs:
Double)

    /// Returns the dot-product of this
vector arithmetic instance with itself.
```

```
    public var magnitudeSquared: Double {  
get }  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Double : VectorArithmetic {  
  
    /// Multiplies each component of this  
    value by the given value.  
    public mutating func scale(by rhs:  
Double)  
  
    /// Returns the dot-product of this  
    vector arithmetic instance with itself.  
    public var magnitudeSquared: Double {  
get }  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension CGFloat : VectorArithmetic {  
  
    /// Multiplies each component of this  
    value by the given value.  
    public mutating func scale(by rhs:  
Double)  
  
    /// Returns the dot-product of this  
    vector arithmetic instance with itself.  
    public var magnitudeSquared: Double {  
get }  
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS  
17.0, watchOS 10.0, *)  
extension Never : ShapeStyle {  
  
    /// The type of shape style this will  
    resolve to.  
    ///  
    /// When you create a custom shape  
    style, Swift infers this type  
    /// from your implementation of the  
    required `resolve` function.  
    public typealias Resolved = Never  
}  
  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension CGPoint {  
  
    public func applying(_ m:  
ProjectionTransform) -> CGPoint  
}  
  
/// Extends `T?` to conform to `Gesture`  
type if `T` also conforms to  
/// `Gesture`. A nil value is mapped to  
an empty (i.e. failing)  
/// gesture.  
@available(iOS 13.0, macOS 10.15, tvOS  
13.0, watchOS 6.0, *)  
extension Optional : Gesture where  
Wrapped : Gesture {
```

```
    /// The type representing the
gesture's value.
    public typealias Value =
Wrapped.Value

    /// The type of gesture representing
the body of `Self`.
    @available(iOS 13.0, tvOS 13.0,
watchOS 6.0, macOS 10.15, *)
    public typealias Body = Never
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Never : Gesture {

    /// The type of value animated by
this keyframes type
    public typealias Value = Never
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension RangeReplaceableCollection
where Self : MutableCollection {

    /// Removes all the elements at the
specified offsets from the collection.
    ///
    /// - Complexity:  $O(*n*)$  where  $*n*$  is
the length of the collection.
    ///
    @available(iOS 13.0, macOS 10.15,
```

```
tvOS 13.0, watchOS 6.0, *)
    public mutating func remove(atOffsets
offsets: IndexSet)
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension MutableCollection {

    /// Moves all the elements at the
    specified offsets to the specified
    /// destination offset, preserving
    ordering.
    ///
    /// Pass an offset as `destination`
    to indicate where in the collection the
    /// moved elements should be
    inserted. Of the elements that are not
    /// represented by the offsets in
    `source`, those before `destination` are
    /// moved toward the beginning of the
    collection to make room for the moved
    /// elements, while those at or after
    `destination` are moved toward the
    /// end.
    ///
    /// In this example, demonstrating
    moving several elements to different
    /// destination offsets,
    `lowercaseOffsets` represents the offsets
    of the
    /// lowercase elements in `letters`:
    ///
```

```
    ///      var letters =
Array("ABcDefgHIJKlmN0")
    ///      let lowercaseOffsets =
IndexSet(...)

    ///      letters.move(fromOffsets:
lowercaseOffsets, toOffset: 2)
    ///      // String(letters) ==
"ABcefglmDHIJKN0"
    ///
    ///      // Reset the `letters` array.
    ///      letters =
Array("ABcDefgHIJKlmN0")
    ///      letters.move(fromOffsets:
lowercaseOffsets, toOffset: 15)
    ///      // String(letters) ==
"ABDHijkN0cefglm"
    ///
    /// If `source` represents a single
element, calling this method with its
    /// own offset, or the offset of the
following element, as `destination`
    /// has no effect.
    ///
    ///      letters =
Array("ABcDefgHIJKlmN0")
    ///      letters.move(fromOffsets:
IndexSet(integer: 2), toOffset: 2)
    ///      // String(letters) ==
"ABcDefgHIJKlmN0"
    ///
    /// - Parameters:
    ///   - source: An index set
representing the offsets of all elements
```

that

```
    ///      should be moved.  
    /// - destination: The offset of  
the element before which to insert the  
    ///      moved elements. `destination`  
must be in the closed range  
    ///      `0...count`.  
    ///  
    /// - Complexity:  $O(*n* \log *n*)$ ,  
where  $*n*$  is the length of the  
collection.
```

```
    @available(iOS 13.0, macOS 10.15,  
tvOS 13.0, watchOS 6.0, *)  
    public mutating func move(fromOffsets  
source: IndexSet, toOffset destination:  
Int)  
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS
```

```
17.0, watchOS 10.0, *)
```

```
extension Double : Animatable {
```

```
    /// The type defining the data to  
animate.
```

```
    @available(iOS 17.0, tvOS 17.0,  
watchOS 10.0, macOS 14.0, *)
```

```
    public typealias AnimatableData =  
Double
```

```
}
```

```
@available(iOS 17.0, macOS 14.0, tvOS
```

```
17.0, watchOS 10.0, *)
```

```
extension CGFloat : Animatable {
```

```
    /// The type defining the data to
    animate.
    @available(iOS 17.0, tvOS 17.0,
    watchOS 10.0, macOS 14.0, *)
    public typealias AnimatableData =
    CGFloat
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Optional : View where Wrapped : View {

    /// The type of view representing the
    body of this view.
    ///
    /// When you create a custom view,
    Swift infers this type from your
    /// implementation of the required
    ``View/body-swift.property`` property.
    public typealias Body = Never
}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension Never : View {

}

@available(iOS 13.0, macOS 10.15, tvOS
13.0, watchOS 6.0, *)
extension CGRect : Animatable {
```

```
    /// The type defining the data to
    /// animate.
    public typealias AnimatableData =
        AnimatablePair<CGPoint.AnimatableData,
        CGSize.AnimatableData>

    /// The data to animate.
    public var animatableData:
        CGRect.AnimatableData
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension FormatStyle where Self ==
SystemFormatStyle.Timer {

    /// Create a timer format style that
    /// counts down in the given interval.
    ///
    /// A timer styled display that
    /// counts down to zero within the given
    /// `interval`.
    ///
    /// - Parameters:
    ///   - interval: The interval during
    ///     which the timer counts down.
    ///   - showsHours: If true, the timer
    ///     shows the hours as a separate element on
    ///     the
    ///     /// formatted string, as long as the
    ///     duration is at least one hour. If
    ///     false, the
    ///     /// timer displays minute values
```

greater than sixty.

    /// - maxFieldCount: The number of fields that can be shown at once. For example,

        /// 1 hour, 34 minutes is shown as `1:34:00` by default, but as `1:34` if the

        /// `maxFieldCount` is set to two. The style automatically excludes more significant

        /// fields if their value is zero and they are not necessary for the format pattern, making

        /// room for less significant fields.

        /// - maxPrecision: The precision at which the input is formatted. E.g. by

        /// default, seconds are shown, making the maximum precision one second. Setting

        /// the maximum precision to `seconds(60)` would only allow hours and minutes to

        /// be shown.

```
public static func
timer(countingDownIn interval:
Range<Date>, showsHours: Bool = true,
maxFieldCount: Int = 3, maxPrecision:
Duration = .seconds(1)) ->
SystemFormatStyle.Timer
```

    /// Create a timer format style that counts up to the given interval.

    ///

```
    /// A timer styled display that
    counts up from zero within the given
    `interval`.
    /// `timerInterval`.
    ///
    /// - Parameters:
    ///   - interval: The interval during
    which the timer counts up.
    ///   - showsHours: If true, the timer
    shows the hours as a separate element
    on the
    /// formatted string, as long as the
    duration is at least one hour. If
    false, the
    /// timer displays minute values
    greater than sixty.
    /// - maxFieldCount: The number of
    fields that can be shown at once. For
    example,
    /// 1 hour, 34 minutes is shown as
    `1:34:00` by default, but as `1:34` if
    the
    /// `maxFieldCount` is set to two.
    The style automatically excludes more
    significant
    /// fields if their value is zero and
    they are not necessary for the format
    pattern, making
    /// room for less significant fields.
    /// - maxPrecision: The precision at
    which the input is formatted. E.g. by
    /// default, seconds are shown,
    making the maximum precision one
```

```
second. Setting
    /// the maximum precision to
` `.seconds(60)` would only allow hours
and minutes to
    /// be shown.
    public static func timer(countingUpIn
interval: Range<Date>, showsHours: Bool =
true, maxFieldCount: Int = 3,
maxPrecision: Duration = .seconds(1)) ->
SystemFormatStyle.Timer
}

@available(iOS 18.0, macOS 15.0, tvOS
18.0, watchOS 11.0, visionOS 2.0, *)
extension FormatStyle where Self ==
SystemFormatStyle.Stopwatch {

    /// Create a stopwatch format style.
    ///
    /// A stopwatch styled display that
    starts counting up from zero at the given
    /// `startDate`.
    ///
    /// - Parameters:
    /// - startDate: The date at which
    the stopwatch starts counting.
    /// - showsHours: If true, the
    stopwatch shows the hours as a separate
    element on
    /// the formatted string, once the
    duration is at least one hour. If
    false, the
    /// stopwatch displays minute values
```

greater than sixty.

    /// - maxFieldCount: The number of fields that can be shown at once. For example,

        /// 1 hour, 34 minutes is shown as `1:34:00` by default, but as `1:34` if the

        /// `maxFieldCount` is set to two. The style automatically excludes more significant

        /// fields if their value is zero and they are not necessary for the format pattern, making

        /// room for less significant fields.

        /// - maxPrecision: The precision at which the input is formatted. E.g. by

        /// default, two fractional digits are shown, making the maximum precision ten

        /// milliseconds. Setting the maximum precision to `seconds(60)` would only allow

        /// hours and minutes to be shown.

```
public static func
stopwatch(startingAt startDate: Date,
showsHours: Bool = true, maxFieldCount:
Int = 4, maxPrecision: Duration
= .milliseconds(10)) ->
SystemFormatStyle.Stopwatch
}
```