SwiftUI / Layout adjustments

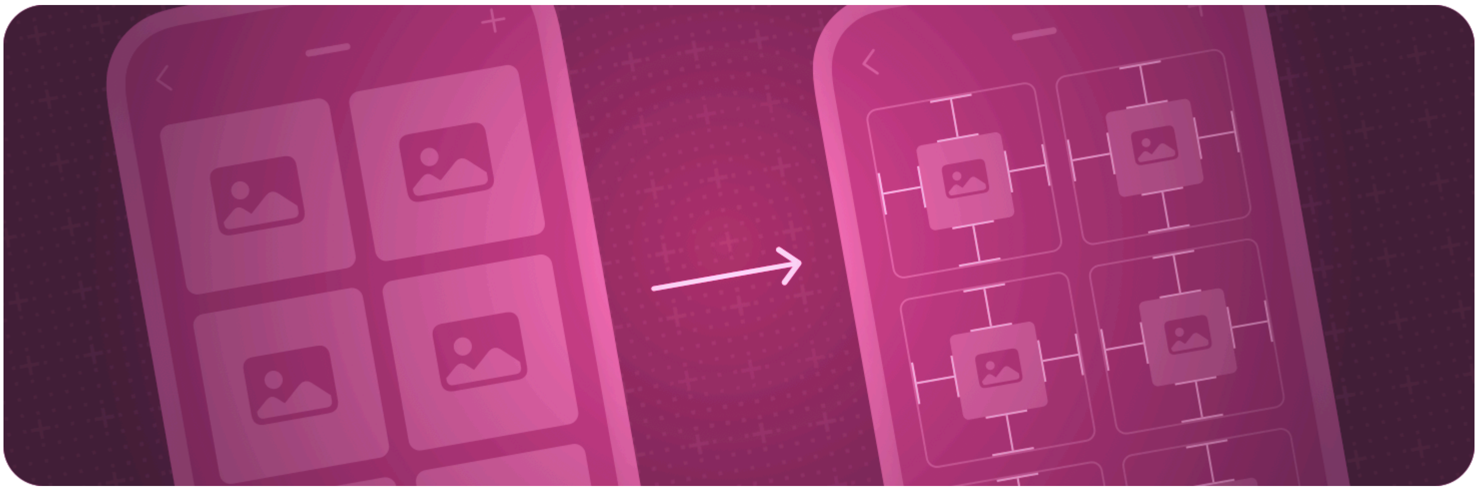
API Collection

Layout adjustments

Make fine adjustments to alignment, spacing, padding, and other layout parameters.

Overview

Layout containers like stacks and grids provide a great starting point for arranging views in your app's user interface. When you need to make fine adjustments, use layout view modifiers. You can adjust or constrain the size, position, and alignment of a view. You can also add padding around a view, and indicate how the view interacts with system-defined safe areas.



To get started with a basic layout, see Layout fundamentals. For design guidance, see Layout in the Human Interface Guidelines.

Topics

Finetuning a layout

Laying out a simple view

Create a view layout by adjusting the size of views.

Inspecting view layout

Determine the position and extent of a view using Xcode previews or by adding temporary borders.

Adding padding around a view

`func padding(_:)`

Adds a different padding amount to each edge of this view.

`func padding(Edge.Set, CGFloat?) -> some View`

Adds an equal padding amount to specific edges of this view.

`func padding3D(_:)`

Pads this view using the edge insets you specify.

`func padding3D(Edge3D.Set, CGFloat?) -> some View`

Pads this view using the edge insets you specify.

`func scenePadding(Edge.Set) -> some View`

Adds padding to the specified edges of this view using an amount that's appropriate for the current scene.

`func scenePadding(ScenePadding, edges: Edge.Set) -> some View`

Adds a specified kind of padding to the specified edges of this view using an amount that's appropriate for the current scene.

`struct ScenePadding`

The padding used to space a view from its containing scene.

Influencing a view's size

`func frame(width: CGFloat?, height: CGFloat?, alignment: Alignment) -> some View`

Positions this view within an invisible frame with the specified size.

```
func frame(depth: CGFloat?, alignment: DepthAlignment) -> some View
```

Positions this view within an invisible frame with the specified depth.

```
func frame(minWidth: CGFloat?, idealWidth: CGFloat?, maxWidth: CGFloat
?, minHeight: CGFloat?, idealHeight: CGFloat?, maxHeight: CGFloat?,
alignment: Alignment) -> some View
```

Positions this view within an invisible frame having the specified size constraints.

```
func frame(minDepth: CGFloat?, idealDepth: CGFloat?, maxDepth: CGFloat
?, alignment: DepthAlignment) -> some View
```

Positions this view within an invisible frame having the specified depth constraints.

```
func containerRelativeFrame(Axis.Set, alignment: Alignment) -> some
View
```

Positions this view within an invisible frame with a size relative to the nearest container.

```
func containerRelativeFrame(Axis.Set, alignment: Alignment, (CGFloat,
Axis) -> CGFloat) -> some View
```

Positions this view within an invisible frame with a size relative to the nearest container.

```
func containerRelativeFrame(Axis.Set, count: Int, span: Int, spacing:
CGFloat, alignment: Alignment) -> some View
```

Positions this view within an invisible frame with a size relative to the nearest container.

```
func fixedSize() -> some View
```

Fixes this view at its ideal size.

```
func fixedSize(horizontal: Bool, vertical: Bool) -> some View
```

Fixes this view at its ideal size in the specified dimensions.

```
func layoutPriority(Double) -> some View
```

Sets the priority by which a parent layout should apportion space to this child.

Adjusting a view’s position

Making fine adjustments to a view’s position

Shift the position of a view by applying the offset or position modifier.

```
func position(CGPoint) -> some View
```

Positions the center of this view at the specified point in its parent’s coordinate space.

```
func position(x: CGFloat, y: CGFloat) -> some View
```

Positions the center of this view at the specified coordinates in its parent's coordinate space.

`func offset(CGSize) -> some View`

Offset this view by the horizontal and vertical amount specified in the offset parameter.

`func offset(x: CGFloat, y: CGFloat) -> some View`

Offset this view by the specified horizontal and vertical distances.

`func offset(z: CGFloat) -> some View`

Brings a view forward in Z by the provided distance in points.

Aligning views

Aligning views within a stack

Position views inside a stack using alignment guides.

Aligning views across stacks

Create a custom alignment and use it to align views across multiple stacks.

`func alignmentGuide(_:computeValue:)`

Sets the view's horizontal alignment.

`struct Alignment`

An alignment in both axes.

`struct HorizontalAlignment`

An alignment position along the horizontal axis.

`struct VerticalAlignment`

An alignment position along the vertical axis.

`struct DepthAlignment`

An alignment position along the depth axis.

`protocol AlignmentID`

A type that you use to create custom alignment guides.

`struct ViewDimensions`

A view's size and alignment guides in its own coordinate space.

`struct ViewDimensions3D`

A view's 3D size and alignment guides in its own coordinate space.

`struct SpatialContainer`

A layout container that aligns overlapping content in 3D space.

Setting margins

`func contentMargins(CGFloat, for: ContentMarginPlacement) -> some View`

Configures the content margin for a provided placement.

`func contentMargins(_:_:for:)`

Configures the content margin for a provided placement.

`struct ContentMarginPlacement`

The placement of margins.

Staying in the safe areas

`func ignoresSafeArea(SafeAreaRegions, edges: Edge.Set) -> some View`

Expands the safe area of a view.

`func safeAreaInset(edge:alignment:spacing:content:)`

Shows the specified content beside the modified view.

`func safeAreaPadding(_:)`

Adds the provided insets into the safe area of this view.

`func safeAreaPadding(Edge.Set, CGFloat?) -> some View`

Adds the provided insets into the safe area of this view.

`struct SafeAreaRegions`

A set of symbolic safe area regions.

Setting a layout direction

`func layoutDirectionBehavior(LayoutDirectionBehavior) -> some View`

Sets the behavior of this view for different layout directions.

`enum LayoutDirectionBehavior`

A description of what should happen when the layout direction changes.

`var layoutDirection: LayoutDirection`

The layout direction associated with the current environment.

`enum LayoutDirection`

A direction in which SwiftUI can lay out content.

`struct LayoutRotationUnaryLayout`

Reacting to interface characteristics

`var isLuminanceReduced: Bool`

A Boolean value that indicates whether the display or environment currently requires reduced luminance.

`var displayScale: CGFloat`

The display scale of this environment.

`var pixelLength: CGFloat`

The size of a pixel on the screen.

`var horizontalSizeClass: UserInterfaceSizeClass?`

The horizontal size class of this environment.

`var verticalSizeClass: UserInterfaceSizeClass?`

The vertical size class of this environment.

`enum UserInterfaceSizeClass`

A set of values that indicate the visual size available to the view.

Accessing edges, regions, and layouts

`enum Edge`

An enumeration to indicate one edge of a rectangle.

`enum Edge3D`

An edge or face of a 3D volume.

`enum HorizontalEdge`

An edge on the horizontal axis.

`enum VerticalEdge`

An edge on the vertical axis.

`struct EdgeInsets`

The inset distances for the sides of a rectangle.

`struct EdgeInsets3D`

The inset distances for the faces of a 3D volume.

See Also

View layout

Layout fundamentals

Arrange views inside built-in layout containers like stacks and grids.

Custom layout

Place views in custom arrangements and create animated transitions between layout types.

Lists

Display a structured, scrollable column of information.

Tables

Display selectable, sortable data arranged in rows and columns.

View groupings

Present views in different kinds of purpose-driven containers, like forms or control groups.

Scroll views

Enable people to scroll to content that doesn’t fit in the current display.