

[SwiftUI](#) / Layout fundamentals

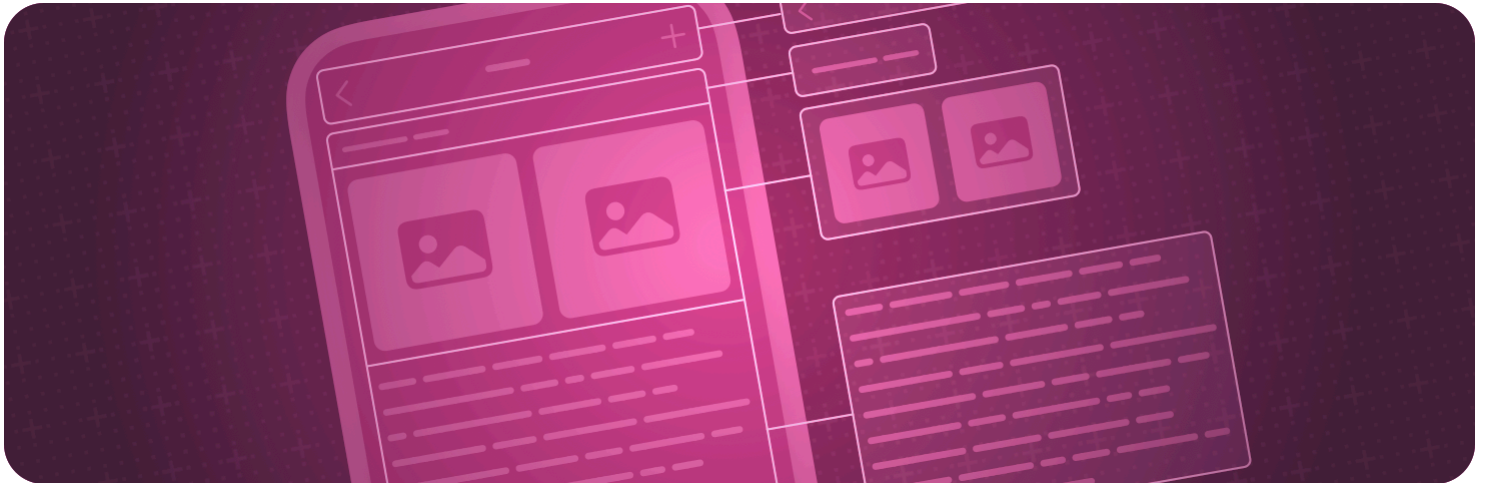
API Collection

Layout fundamentals

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

Overview

Use layout containers to arrange the elements of your user interface. Stacks and grids update and adjust the positions of the subviews they contain in response to changes in content or interface dimensions. You can nest layout containers inside other layout containers to any depth to achieve complex layout effects.



To finetune the position, alignment, and other elements of a layout that you build with layout container views, see [Layout adjustments](#). To define custom layout containers, see [Custom layout](#). For design guidance, see [Layout](#) in the Human Interface Guidelines.

Topics

Choosing a layout

- 📄 Picking container views for your content

Build flexible user interfaces by using stacks, grids, lists, and forms.

Statically arranging views in one dimension

- 📄 Building layouts with stack views

Compose complex layouts from primitive container views.

`struct HStack`

A view that arranges its subviews in a horizontal line.

`struct VStack`

A view that arranges its subviews in a vertical line.

Dynamically arranging views in one dimension

- 📄 Grouping data with lazy stack views

Split content into logical sections inside lazy stack views.

- 📄 Creating performant scrollable stacks

Display large numbers of repeated views efficiently with scroll views, stack views, and lazy stacks.

`struct LazyHStack`

A view that arranges its children in a line that grows horizontally, creating items only as needed.

`struct LazyVStack`

A view that arranges its children in a line that grows vertically, creating items only as needed.

`struct PinnedScrollableViews`

A set of view types that may be pinned to the bounds of a scroll view.

Statically arranging views in two dimensions

`struct Grid`

A container view that arranges other views in a two dimensional layout.

```
struct GridRow
```

A horizontal row in a two dimensional grid container.

```
func gridCellColumns(Int) -> some View
```

Tells a view that acts as a cell in a grid to span the specified number of columns.

```
func gridCellAnchor(UnitPoint) -> some View
```

Specifies a custom alignment anchor for a view that acts as a grid cell.

```
func gridCellUnsizeAxes(Axis.Set) -> some View
```

Asks grid layouts not to offer the view extra size in the specified axes.

```
func gridColumnAlignment(HorizontalAlignment) -> some View
```

Overrides the default horizontal alignment of the grid column that the view appears in.

Dynamically arranging views in two dimensions

```
struct LazyHGrid
```

A container view that arranges its child views in a grid that grows horizontally, creating items only as needed.

```
struct LazyVGrid
```

A container view that arranges its child views in a grid that grows vertically, creating items only as needed.

```
struct GridItem
```

A description of a row or a column in a lazy grid.

Layering views

 Adding a background to your view

Compose a background behind your view and extend it beyond the safe area insets.

```
struct ZStack
```

A view that overlays its subviews, aligning them in both axes.

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

Controls the display order of overlapping views.

```
func background<V>(alignment: Alignment, content: () -> V) -> some View
```

Layers the views that you specify behind this view.

```
func background<S>(S, ignoresSafeAreaEdges: Edge.Set) -> some View
```

Sets the view's background to a style.

```
func background(ignoresSafeAreaEdges: Edge.Set) -> some View
```

Sets the view's background to the default background style.

```
func background(_:in:fillStyle:)
```

Sets the view's background to an insettable shape filled with a style.

```
func background(in:fillStyle:)
```

Sets the view's background to an insettable shape filled with the default background style.

```
func overlay<V>(alignment: Alignment, content: () -> V) -> some View
```

Layers the views that you specify in front of this view.

```
func overlay<S>(S, ignoresSafeAreaEdges: Edge.Set) -> some View
```

Layers the specified style in front of this view.

```
func overlay<S, T>(S, in: T, fillStyle: FillStyle) -> some View
```

Layers a shape that you specify in front of this view.

```
var backgroundMaterial: Material?
```

The material underneath the current view.

```
func containerBackground<S>(S, for: ContainerBackgroundPlacement) ->  
some View
```

Sets the container background of the enclosing container using a view.

```
func containerBackground<V>(for: ContainerBackgroundPlacement,  
alignment: Alignment, content: () -> V) -> some View
```

Sets the container background of the enclosing container using a view.

```
struct ContainerBackgroundPlacement
```

The placement of a container background.

Automatically choosing the layout that fits

```
struct ViewThatFits
```

A view that adapts to the available space by providing the first child view that fits.

Separators

`struct Spacer`

A flexible space that expands along the major axis of its containing stack layout, or on both axes if not contained in a stack.

`struct Divider`

A visual element that can be used to separate other content.

See Also

View layout

☰ Layout adjustments

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

☰ 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.