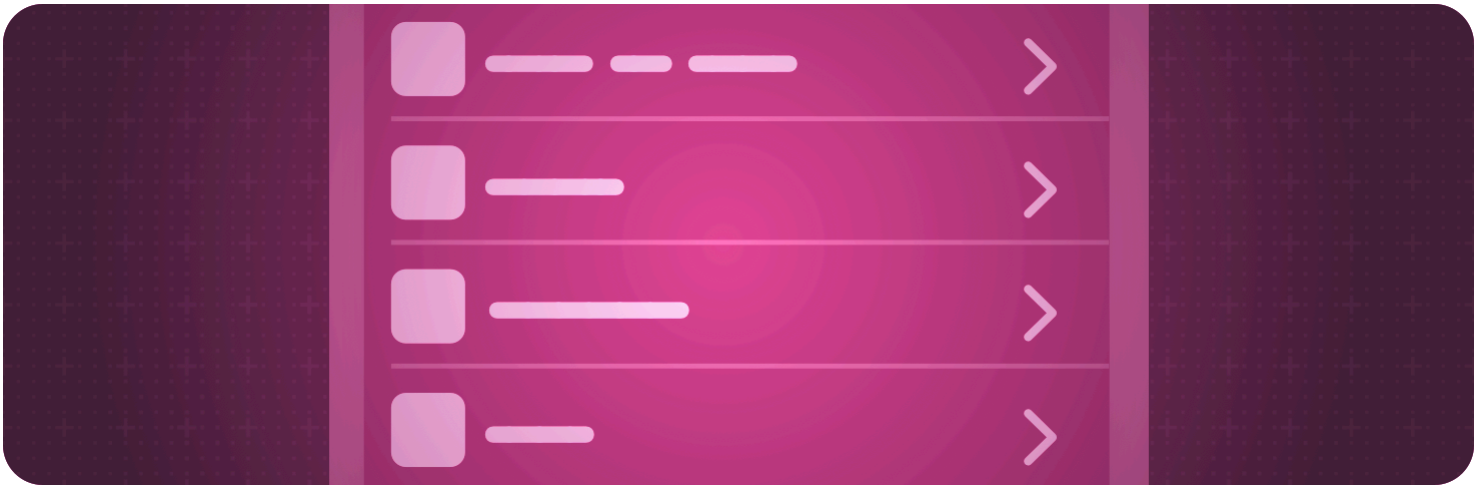
API Collection

# Lists

Display a structured, scrollable column of information.

## Overview

Use a list to display a one-dimensional vertical collection of views.



The list is a complex container type that automatically provides scrolling when it grows too large for the current display. You build a list by providing it with individual views for the rows in the list, or by using a `ForEach` to enumerate a group of rows. You can also mix these strategies, blending any number of individual views and `ForEach` constructs.

Use view modifiers to configure the appearance and behavior of a list and its rows, headers, sections, and separators. For example, you can apply a style to the list, add swipe gestures to individual rows, or make the list refreshable with a pull-down gesture. You can also use the configuration associated with Scroll views to control the list's implicit scrolling behavior.

For design guidance, see Lists and tables in the Human Interface Guidelines.

# Topics

## Creating a list

Displaying data in lists

Visualize collections of data with platform-appropriate appearance.

`struct List`

A container that presents rows of data arranged in a single column, optionally providing the ability to select one or more members.

`func listStyle<S>(S) -> some View`

Sets the style for lists within this view.

## Disclosing information progressively

`struct OutlineGroup`

A structure that computes views and disclosure groups on demand from an underlying collection of tree-structured, identified data.

`struct DisclosureGroup`

A view that shows or hides another content view, based on the state of a disclosure control.

`func disclosureGroupStyle<S>(S) -> some View`

Sets the style for disclosure groups within this view.

## Configuring a list's layout

`func listRowInsets(EdgeInsets?) -> some View`

Applies an inset to the rows in a list.

`var defaultMinListRowHeight: CGFloat`

The default minimum height of rows in a list.

`var defaultMinListHeaderHeight: CGFloat?`

The default minimum height of a header in a list.

`func listRowSpacing(CGFloat?) -> some View`

Sets the vertical spacing between two adjacent rows in a List.

`func listSectionSpacing(_:)`

Sets the spacing between adjacent sections in a `List` to a custom value.

`struct ListSectionSpacing`

The spacing options between two adjacent sections in a list.

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

Set the section margins for the specific edges.

## Configuring rows

`func listItemTint(_:)`

Sets a fixed tint color for content in a list.

`struct ListItemTint`

A tint effect configuration that you can apply to content in a list.

## Configuring headers

`func headerProminence(Prominence) -> some View`

Sets the header prominence for this view.

`var headerProminence: Prominence`

The prominence to apply to section headers within a view.

`enum Prominence`

A type indicating the prominence of a view hierarchy.

## Configuring separators

`func listRowSeparatorTint(Color?, edges: VerticalEdge.Set) -> some View`

Sets the tint color associated with a row.

`func listSectionSeparatorTint(Color?, edges: VerticalEdge.Set) -> some View`

Sets the tint color associated with a section.

`func listRowSeparator(Visibility, edges: VerticalEdge.Set) -> some View`

Sets the display mode for the separator associated with this specific row.

`func listSectionSeparator(Visibility, edges: VerticalEdge.Set) -> some View`

Sets whether to hide the separator associated with a list section.

## Configuring backgrounds

`func listRowBackground<V>(V?) -> some View`

Places a custom background view behind a list row item.

`func alternatingRowBackgrounds(AlternatingRowBackgroundBehavior) -> some View`

Overrides whether lists and tables in this view have alternating row backgrounds.

`struct AlternatingRowBackgroundBehavior`

The styling of views with respect to alternating row backgrounds.

`var backgroundProminence: BackgroundProminence`

The prominence of the background underneath views associated with this environment.

`struct BackgroundProminence`

The prominence of backgrounds underneath other views.

## Displaying a badge on a list item

`func badge(_:)`

Generates a badge for the view from an integer value.

`func badgeProminence(BadgeProminence) -> some View`

Specifies the prominence of badges created by this view.

`var badgeProminence: BadgeProminence`

The prominence to apply to badges associated with this environment.

`struct BadgeProminence`

The visual prominence of a badge.

## Configuring interaction

`func swipeActions<T>(edge: HorizontalEdge, allowsFullSwipe: Bool, content: () -> T) -> some View`

Adds custom swipe actions to a row in a list.

`func selectionDisabled(Bool) -> some View`

Adds a condition that controls whether users can select this view.

`func listRowHoverEffect(HoverEffect?) -> some View`

Requests that the containing list row use the provided hover effect.

`func listRowHoverEffectDisabled(Bool) -> some View`

Requests that the containing list row have its hover effect disabled.

## Refreshing a list's content

`func refreshable(action: () async -> Void) -> some View`

Marks this view as refreshable.

`var refresh: RefreshAction?`

A refresh action stored in a view's environment.

`struct RefreshAction`

An action that initiates a refresh operation.

## Editing a list

`func moveDisabled(Bool) -> some View`

Adds a condition for whether the view's view hierarchy is movable.

`func deleteDisabled(Bool) -> some View`

Adds a condition for whether the view's view hierarchy is deletable.

`var editMode: Binding<EditMode>?`

An indication of whether the user can edit the contents of a view associated with this environment.

`enum EditMode`

A mode that indicates whether the user can edit a view's content.

`struct EditActions`

A set of edit actions on a collection of data that a view can offer to a user.

`struct EditableCollectionContent`

An opaque wrapper view that adds editing capabilities to a row in a list.

`struct IndexedIdentifierCollection`

A collection wrapper that iterates over the indices and identifiers of a collection together.

### Configuring a section index

`func listSectionIndexVisibility(Visibility) -> some View`

Changes the visibility of the list section index.

`func sectionIndexLabel(_:)`

Sets the label that is used in a section index to point to this section, typically only a single character long.

---

# See Also

### View layout

- Layout fundamentals

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

- 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.

- 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.