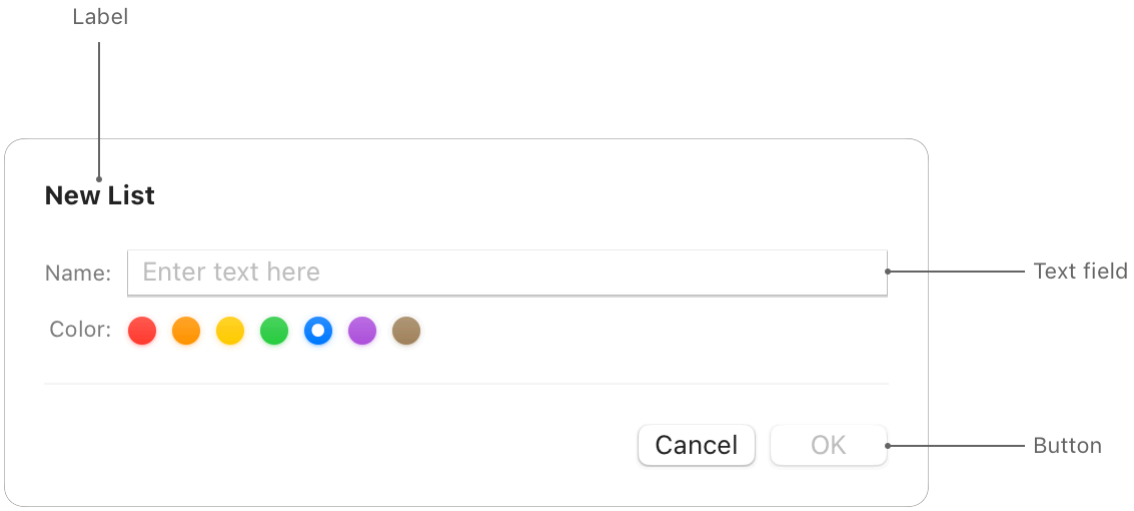
API Collection

# Views and Controls

Present your content onscreen and handle user input and events.

## Overview

Views and controls are the building blocks of your app's user interface.




Views can host other views. Embedding one view inside another creates a containment relationship between the host view (known as the *superview*) and the embedded view (known as the *subview*). View hierarchies make it easier to manage views.

You can also use views to do any of the following:

- Respond to touches and other events (either directly or in coordination with gesture recognizers).
- Draw custom content using Core Graphics.
- Respond to focus changes.
- Animate the size, position, and appearance attributes of the view using Core Animation.

Favor AppKit views and controls whenever possible. These components adapt automatically to system changes, and many support appearance customizations to support the look and feel you want in your app. When AppKit doesn't provide the exact view or control you need, you can create a custom view.

`NSView` is the root class for all views and defines their common behavior. `NSControl` defines additional behaviors that are specific to buttons, switches, and other views designed for user interactions.

For additional information about how to use views and controls, see Human Interface Guidelines.

# Topics

## View fundamentals

`class NSView`

The infrastructure for drawing, printing, and handling events in an app.

`class NSControl`

A specialized view, such as a button or text field, that notifies your app of relevant events using the target-action design pattern.

`class NSCell`

A mechanism for displaying text or images in a view object without the overhead of a full `NSView` subclass.

`class NSActionCell`

An active area inside a control.

## Container views

Use container views to arrange the views of your interface and to facilitate navigation among those views.

{} Localization-friendly layouts in macOS

This project demonstrates localization-friendly auto layout constraints.

Grid View

Arrange views in a flexible grid, and handle the layout associated with those views.

`class NSSplitView`

A view that arranges two or more views in a linear stack running horizontally or vertically.

{} Organize Your User Interface with a Stack View

Group individual views in your app's user interface into a scrollable stack view.

`class NSStackView`

A view that arranges an array of views horizontally or vertically and updates their placement and sizing when the window size changes.

`class NSTabView`

A multipage interface that displays one page at a time.

Scroll View

Provide an interface for navigating content that is too large to fit in the available space.

## Content views

Use content views to organize and display your app's data.

Browser View

Provide a column-based interface for viewing and navigating hierarchical information.

Collection View

Display one or more subviews in a highly configurable arrangement.

Outline View

Display a list-based interface for hierarchical data, where each level of hierarchy is indented from the previous one.

Table View

Display custom data in rows and columns.

`class NSTextView`

A view that draws text and handles user interactions with that text.

## Controls

Use controls to handle specific types of user interactions. Controls are specialized views that use the target-action design pattern to notify your app of interactions with their content.

Responding to control-based events using target-action

Handle user input by connecting buttons, sliders, and other controls to your app's code using the target-action design pattern.

`class` `NSButton`

A control that defines an area on the screen that a user clicks to trigger an action.

`class` `NSColorWell`

A control that displays a color value and lets the user change that color value.

Combo Box

Display a list of values in a pop-up menu that lets the user select a value or type in a custom value.

`class` `NSComboButton`

A button with a pull-down menu and a default action.

Date Picker

Display a calendar date and provide controls for editing the date value.

`class` `NSImageView`

A display of image data in a frame.

`class` `NSLevelIndicator`

A visual representation of a level or quantity, using discrete values.

Path Control

A display of a file system path or virtual path information.

`class` `NSPopUpButton`

A control for selecting an item from a list.

`class` `NSProgressIndicator`

An interface that provides visual feedback to the user about the status of an ongoing task.

`class` `NSRuleEditor`

An interface for configuring a rule-based list of options.

`class` `NSPredicateEditor`

A defined set of rules that allows the editing of predicate objects.

Search Field

Provide a text field that is optimized for text-based search interfaces.

`class` `NSSegmentedControl`

Display one or more buttons in a single horizontal group.

Slider

Display a range of values from which the user selects a single value.

`class NSStepper`

An interface with up and down arrow buttons for incrementing or decrementing a value.

Text Field

Provide a simple interface for displaying and editing text, including support for password fields and secure forms of text entry.

Token Field

Provide a text field whose text can be rendered in a visually distinct way so that users can recognize portions more easily.

Toolbar

Provide a space for controls under a window's title bar and above your custom content.

`class NSSwitch`

A control that offers a binary choice.

`class NSMatrix`

A legacy interface for grouping radio buttons or other types of cells together.

## Liquid Glass effects

`class NSGlassEffectView`

A view that embeds its content view in a dynamic glass effect.

`class NSGlassEffectContainerView`

A view that efficiently merges descendant glass effect views together when they are within a specified proximity to each other.

## Interacting with adjacent views

`class NSBackgroundExtensionView`

A view that extends content to fill its own bounds.

## Visual adornments

Add purely decorative elements to your user interface.

`class` NSVisualEffectView

A view that adds translucency and vibrancy effects to the views in your interface.

`class` NSBox

A stylized rectangular box with an optional title.

## UI validation

`protocol` NSUserInterfaceValidations

A protocol that a custom class can adopt to manage the enabled state of a UI element.

`protocol` NSValidatedUserInterfaceItem

A protocol that a custom class can adopt to manage the automatic enablement of a UI control.

## Tool tips

`protocol` NSViewToolTipOwner

A set of methods for dynamically associating a tool tip with a view.

## Related types

`enum` NSRectAlignment

Constants that specify alignment to an edge or a set of edges depending on the user interface layout direction.

`struct` NSDirectionalEdgeInsets

The inset distances for views, taking the user interface layout direction into account.

`struct` NSDirectionalRectEdge

---

# See Also

## User Interface

View Management

Manage your user interface, including the size and position of views in a window.

- View Layout

  Position and size views using a stack view or Auto Layout constraints.

- Appearance Customization

  Add Dark Mode support to your app, and use appearance proxies to modify your UI.

- Animation

  Animate your views and other content to create a more engaging experience for users.

- Windows, Panels, and Screens

  Organize your view hierarchies and facilitate their display onscreen.

- Sound, Speech, and Haptics

  Play sounds and haptic feedback, and incorporate speech recognition and synthesis into your interface.

- Supporting Continuity Camera in Your Mac App

  Incorporate scanned documents and pictures from a user's iPhone, iPad, or iPod touch into your Mac app using Continuity Camera.