Sample Code

# Adjusting your layout with keyboard layout guide

**Respond dynamically to keyboard movement by using the tracking features of the keyboard layout guide.**

Download

iOS 15.0+ | iPadOS 15.0+ | Xcode 13.0+

## Overview

When your app presents a keyboard, you want it to integrate well into your layout. This sample code project demonstrates how to configure constraints when you set `followsUndockedKeyboard` to `true` on the `keyboardLayoutGuide` property in `UIView` to adapt your layout to the movement of the floating keyboard onscreen. The sample illustrates several concepts, such as handling when the keyboard is close to the top of the screen, and adapting the layout when the additional views don't have constraints with a direct relationship to the `keyboardLayoutGuide`.

When you use `keyboardLayoutGuide` and leave `followsUndockedKeyboard` set to the default value of `false`, the guide matches the keyboard when it docks. When the keyboard isn't onscreen, the guide is at the bottom of the window and has a height equal to the bottom of the current `safeAreaInsets`. By default, when the keyboard undocks, the guide behaves the same as when you dismiss the keyboard or the keyboard isn't visible. You can use the guide like any other layout guide.

```
view.keyboardLayoutGuide.topAnchor.constraint(
    equalToSystemSpacingBelow: textView.bottomAnchor, multiplier: 1.0).isActive = tr
```

See `UILayoutGuide` for more information about layout guides.

## Follow the undocked keyboard

For more precise layout responses to the undocked keyboard, the sample sets `followsUndockedKeyboard` to `true` and then creates tracking constraints that activate and deactivate when the keyboard approaches or leaves an edge. Tracking constraints only apply when `followsUndockedKeyboard` is `true`.

```
// This is necessary to allow the various edge constraints to engage.
view.keyboardLayoutGuide.followsUndockedKeyboard = true
```

## Enable tracking constraints

The sample shows how the `UITrackingLayoutGuide` activates or deactivates constraints when the `keyboardLayoutGuide` approaches or leaves an edge. The sample code passes an array of constraints to the `keyboardLayoutGuide`, and indicates which edges trigger the change. The sample configures an array of constraints to activate when `awayFrom` the leading and trailing edges, so the constraints activate when the keyboard docks or when the floating keyboard is in the middle area of the screen. Conversely, the constraints deactivate when the floating keyboard approaches either the leading or trailing edge.

> **Important**
>
> Constraints that use the keyboard layout guide's anchors directly don't activate and deactivate automatically. To get the full tracking behavior, the sample sets constraints using the `setConstraints(_:activeWhenNearEdge:)` and `setConstraints(_:activeWhenAwayFrom:)` methods on `UITrackingLayoutGuide`.

The following example shows how to pin a view to the top of the `keyboardLayoutGuide` when the guide is `awayFrom` the top edge, and to the bottom of the view's `safeAreaLayoutGuide` when it's `near` the top edge:

```
// When the keyboard isn't near the top, tie the edit view to the keyboard layout gu
// deactivates when near the top).
let editViewOnKeyboard = view.keyboardLayoutGuide.topAnchor.constraint(equalTo: edit
editViewOnKeyboard.identifier = "editViewOnKeyboard"
view.keyboardLayoutGuide.setConstraints([editViewOnKeyboard], activeWhenAwayFrom: .t

// When the keyboard is near the top, tie the edit view to the bottom anchor of the
let editViewOnBottom = view.safeAreaLayoutGuide.bottomAnchor.constraint(equalTo: edi
editViewOnBottom.identifier = "editViewOnBottom"
```

The sample demonstrates that it isn't necessary to pin views to the `keyboardLayoutGuide` to affect their constraints when the keyboard approaches an edge. The guide can take any array of constraints in the layout and activate and deactivate them when necessary. For example, the following code shows an `imageView` with no constraint relationships to the `keyboardLayoutGuide` itself, but the image moves to the opposite side of the screen from the keyboard to stay visible as the keyboard moves around:

```
let centeredImage = imageView.centerXAnchor.constraint(equalTo: view.centerXAnchor)
centeredImage.identifier = "centeredImage"
view.keyboardLayoutGuide.setConstraints([centeredImage], activeWhenAwayFrom: [.leadi

let imageViewToLeading = imageView.leadingAnchor.constraint(
    equalToSystemSpacingAfter: view.safeAreaLayoutGuide.leadingAnchor, multiplier: 1
imageViewToLeading.identifier = "imageViewToLeading"

let nearTrailingConstraints = [ editViewToUndockedKeyboardTrailing, imageViewToLeadi
view.keyboardLayoutGuide.setConstraints(nearTrailingConstraints, activeWhenNearEdge:

let imageViewToTrailing = view.safeAreaLayoutGuide.trailingAnchor.constraint(
    equalToSystemSpacingAfter: imageView.trailingAnchor, multiplier: 1.0)
imageViewToTrailing.identifier = "imageViewToTrailing"

let nearLeadingConstraints = [ editViewToKeyboardLeading, imageViewToTrailing ]
view.keyboardLayoutGuide.setConstraints(nearLeadingConstraints, activeWhenNearEdge:
```

## Account for keyboard types

The following list describes which edges the system reports the `keyboardLayoutGuide` is `near` and `awayFrom` when different types of keyboards are active:

- Docked keyboards:
  - Are always `awayFrom` leading, trailing, and top edges
  - Are always `near` the bottom edge
- Split and undocked keyboards:
  - Are always `awayFrom` leading, trailing, and bottom edges
  - Can be `near` the top edge
- Floating keyboards:

- Can be `awayFrom` all edges
- Can be `near` any edge or any two adjacent edges

- The shortcuts bar (available with an external attached keyboard):
  - Is always `awayFrom` the top edge and `near` the bottom edge
  - Can be `near` the leading or trailing edge when in a collapsed state

> **Note**
>
> Although the floating keyboard can be `near` two adjacent edges at once, the keyboard doesn't support any scenarios where it is `near` more than two edges, and is most frequently `near` only one. It can be `awayFrom` any combination of edges, however. If the floating keyboard is available, but not over the app, the `keyboardLayoutGuide` behaves as if the keyboard is in a dismissed state.

When the app is in 1/3 split view mode, the `keyboardLayoutGuide` is `awayFrom` leading and trailing edges, `near` the bottom edge if docked, and can be `near` the top edge if floating or undocked.

## Handle edge combinations

The sample shows constraints that are active when `near` or `awayFrom` multiple edges. The system activates or deactivates the constraints only when the position of the keyboard meets all edge requirements. Because a docked keyboard is `awayFrom` leading and trailing edges, but `near` the bottom edge, the constraint in the example below only activates when the keyboard undocks and is `awayFrom` the leading and trailing edges. This situation happens with undocked full or split keyboards, and with a floating keyboard in the horizontal middle of the screen.

```
let editCenterXToKeyboard = view.keyboardLayoutGuide.centerXAnchor.constraint(equalT
editCenterXToKeyboard.identifier = "editCenterXToKeyboard"

view.keyboardLayoutGuide.setConstraints([editCenterXToKeyboard], activeWhenAwayFrom:
```

# See Also

## Keyboard layout

`class UIKeyboardLayoutGuide`

A layout guide that represents the space the keyboard occupies in your app's layout.

`class` `UITrackingLayoutGuide`

A layout guide that automatically activates and deactivates layout constraints depending on its proximity to edges.