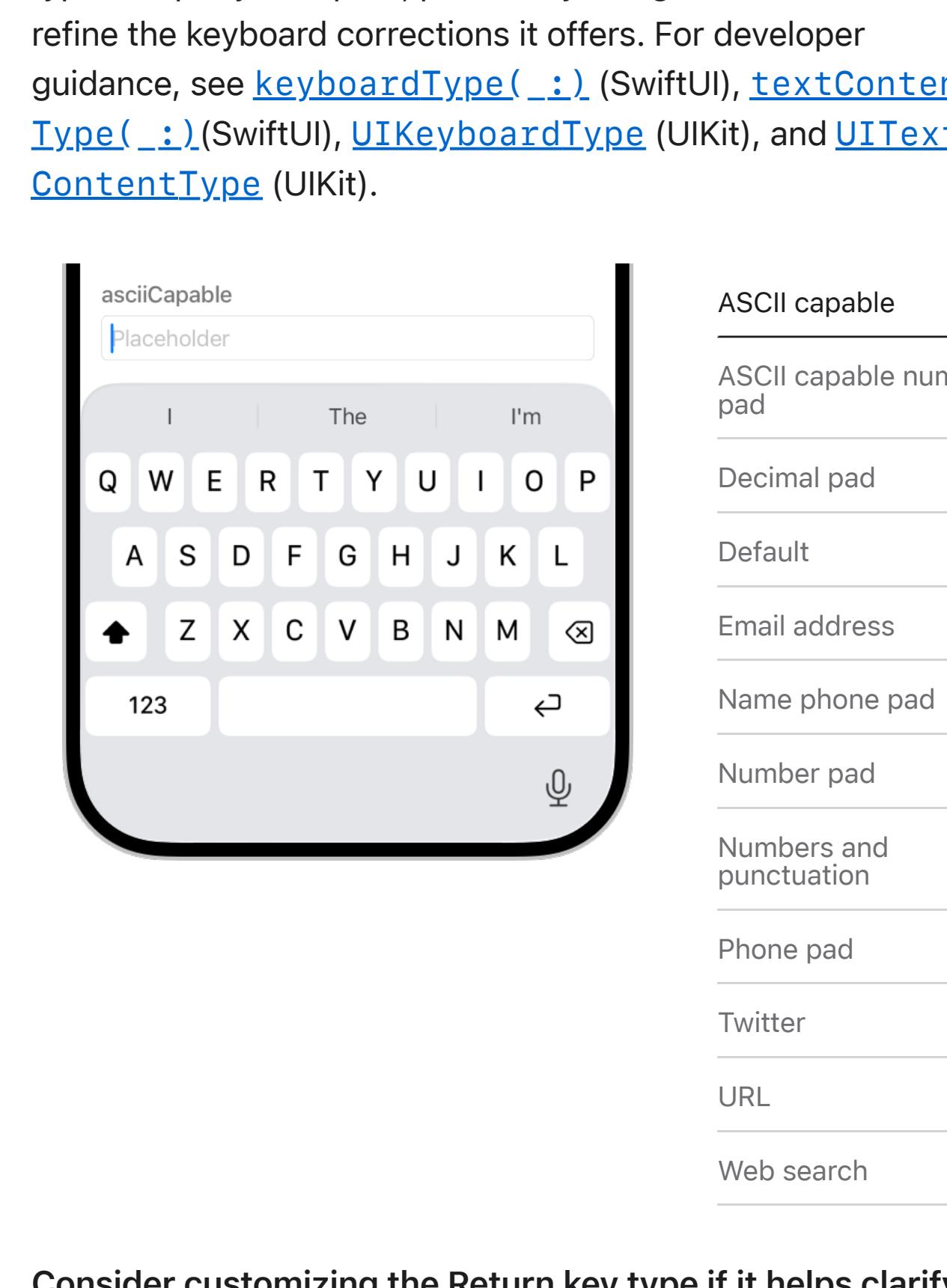


Virtual keyboards

On devices without physical keyboards, the system offers various types of virtual keyboards people can use to enter data.



Supported platforms



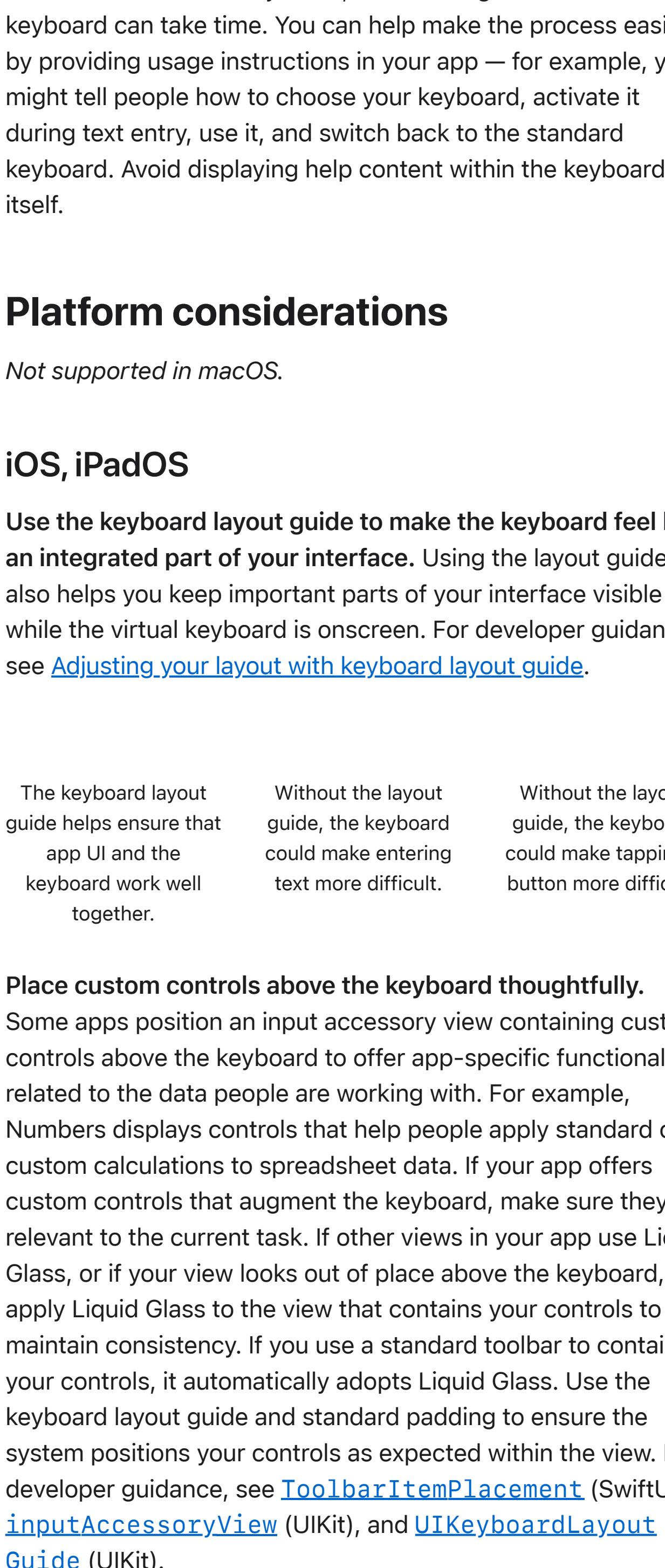
- Virtual keyboards
- Best practices
- Custom input views
- Custom keyboards
- Platform considerations
- Resources
- Change log

A virtual keyboard can provide a specific set of keys that are optimized for the current task; for example, a keyboard that supports entering email addresses can include the "@" character and a period or even ".com". A virtual keyboard doesn't support keyboard shortcuts.

When it makes sense in your app, you can replace the system-provided keyboard with a custom view that supports app-specific data entry. In iOS, iPadOS, and tvOS, you can also create an app extension that offers a custom keyboard people can install and use in place of the standard keyboard.

Best practices

Choose a keyboard that matches the type of content people are editing. For example, you can help people enter numeric data by providing the numbers and punctuation keyboard. When you specify a semantic meaning for a text input area, the system can automatically provide a keyboard that matches the type of input you expect, potentially using this information to refine the keyboard corrections it offers. For developer guidance, see `keyboardType(_ :)` (SwiftUI), `textContentType(_ :)` (SwiftUI), `UIKeyboardType` (UIKit), and `UITextContentType` (UIKit).



Consider customizing the Return key type if it helps clarify the text-entry experience. The Return key type is based on the keyboard type you choose, but you can change this if it makes sense in your app. For example, if your app initiates a search, you can use a search Return key type rather than the standard one so the experience is consistent with other places people initiate search. For developer guidance, see `submitLabel(_ :)` (SwiftUI) and `UIReturnKeyType` (UIKit).

Custom input views

In some cases, you can create an *input* view if you want to provide custom functionality that enhances data-entry tasks in your app. For example, Numbers provides a custom input view for entering numeric values while editing a spreadsheet. A custom input view replaces the system-provided keyboard while people are in your app. For developer guidance, see `ToolbarItemPlacement` (SwiftUI) and `inputViewController` (UIKit).

Make sure your custom input view makes sense in the context of your app. In addition to making data entry simple and intuitive, you want people to understand the benefits of using your custom input view. Otherwise, they may wonder why they can't regain the system keyboard while in your app.

Play the standard keyboard sound while people type. The keyboard sound provides familiar feedback when people tap a key on the system keyboard, so they're likely to expect the same sound when they tap keys in your custom input view. People can turn keyboard sounds off for all keyboard interactions in Settings > Sounds. For developer guidance, see `playInputClick(_ :)` (UIKit).

Custom keyboards

In iOS, iPadOS, and tvOS, you can provide a custom keyboard that replaces the system keyboard by creating an app extension. An *app extension* is code you provide that people can install and use to extend the functionality of a specific area of the system; to learn more, see [App extensions](#).

After people choose your custom keyboard in Settings, they can use it for text entry within any app, except when editing secure text fields and phone number fields. People can choose multiple custom keyboards and switch between them at any time. For developer guidance, see [Creating a custom keyboard](#).

Custom keyboards make sense when you want to expose unique keyboard functionality systemwide, such as a novel way of inputting text or the ability to type in a language the system doesn't support. If you want to provide a custom keyboard for people to use only while they're in your app, consider creating a custom input view instead.

Provide an obvious and easy way to switch between keyboards. People know that the Globe key on the standard keyboard — which replaces the dedicated Emoji key when multiple keyboards are available — quickly switches to other keyboards, and they expect a similarly intuitive experience in your keyboard.

Avoid duplicating system-provided keyboard features. On some devices, the Emoji/Globe key and Dictation key

automatically appear beneath the keyboard, even when people are using custom keyboards. Your app can't affect these keys, and it's likely to be confusing if you repeat them in your keyboard.

Consider providing a keyboard tutorial in your app. People are used to the standard keyboard, and learning how to use a new keyboard can take time. You can help make the process easier by providing usage instructions in your app — for example, you might tell people how to choose your keyboard, activate it during text entry, use it, and switch back to the standard keyboard. Avoid displaying help content within the keyboard itself.

Platform considerations

Not supported in macOS.

iOS, iPadOS

Use the keyboard layout guide to make the keyboard feel like an integrated part of your interface. Using the layout guide also helps you keep important parts of your interface visible while the virtual keyboard is onscreen. For developer guidance, see [Adjusting your layout with keyboard layout guide](#).

The keyboard layout guide helps ensure that app UI and the keyboard work well together.

Without the layout guide, the keyboard could make entering text more difficult.

Without the layout guide, the keyboard could make tapping a button more difficult.

Place custom controls above the keyboard thoughtfully. Some apps position an input accessory view containing custom controls above the keyboard to offer app-specific functionality related to the data people are working with. For example, Numbers displays controls that help people apply standard or custom calculations to spreadsheet data. If your app offers custom controls that augment the keyboard, make sure they're relevant to the current task. If other views in your app use Liquid Glass, or if your view looks out of place above the keyboard, apply Liquid Glass to the view that contains your controls to maintain consistency. If you use a standard toolbar to contain your controls, it automatically adopts Liquid Glass. Use the keyboard layout guide and standard padding to ensure the system positions your controls as expected within the view. For developer guidance, see `ToolbarItemPlacement` (SwiftUI), `inputAccessoryView` (UIKit), and `UIKeyboardLayoutGuide` (UIKit).

tvOS

tvOS displays a linear virtual keyboard when people select a text field using the Siri Remote.

Note
A grid keyboard screen appears when people use devices other than the Siri Remote, and the layout of content automatically adapts to the keyboard.

When people activate a digit entry view, tvOS displays a digit-specific keyboard. For guidance, see [Digit entry views](#).

watchOS

On Apple Watch, a text field can show a keyboard if the device screen is large enough. Otherwise, the system lets people use dictation or Scribble to enter information. You can't change the keyboard type in watchOS, but you can set the content type of the text field. The system uses this information to make text entry easier, such as by offering suggestions. For developer guidance, see `textContentType(_ :)` (SwiftUI).

People can also use a nearby paired iPhone to enter text on Apple Watch.

Resources

Related

[Entering data](#)

[Keyboards](#)

[Layout](#)

Developer documentation

[keyboardType\(_ :\)](#) — SwiftUI

[textContentType\(_ :\)](#) — SwiftUI

[UIKeyboardType](#) — UIKit

Topics & Technologies

Accessibility

Downloads

Forums

System Status

Videos

Apple Developer Connect

Certificates, IDs, & Profiles

Feedback Assistant

Resources

Documentation

Tutorials

Downloads

Forums

Videos

Apple Developer Connect

Certificates, IDs, & Profiles

Feedback Assistant

Programs

Apple Developer Program

Apple Developer Enterprise Program

App Store Small Business Program

MFN Program

News Partner Program

Video Partner Program

Security Bounty Program

Security Research Device Program

Events

Meet with Apple

Apple Developer Centers

App Store Awards

Apple Design Awards

Apple Developer Academies

WWDC

Change log

Date	Changes
June 9, 2025	Added guidance for displaying custom controls above the keyboard, and updated to reflect virtual keyboard availability in watchOS.
February 2, 2024	Clarified the virtual keyboard's support for direct and indirect gestures in visionOS.
December 5, 2023	Added artwork for visionOS.
June 21, 2023	Changed page title from Onscreen keyboards and updated to include guidance for visionOS.

Added guidance for displaying custom controls above the keyboard, and updated to reflect virtual keyboard availability in watchOS.

Clarified the virtual keyboard's support for direct and indirect gestures in visionOS.

Added artwork for visionOS.

Changed page title from Onscreen keyboards and updated to include guidance for visionOS.

Platforms

iOS

iPadOS

macOS

tvOS

visionOS

watchOS

Tools

Swift

SwiftUI

TestFlight

Xcode

Xcode Cloud

SF Symbols

Topics & Technologies

Accessibility

Annotations

App Extension

App Store

Audio & Video

Augmented Reality

Design

Distribution

Education

Fonts

Games

Health & Fitness

In-App Purchase

Localization

Maps & Location

Machine Learning & AI

Open Source

Security

Safari & Web

Resources

Documentation

Tutorials

Downloads

Forums

Videos

Apple Developer Connect

Certificates, IDs, & Profiles

Feedback Assistant

Programs

Apple Developer Program

Apple Developer Enterprise Program

App Store Small Business Program

MFN Program

News Partner Program

Video Partner Program

Security Bounty Program

Security Research Device Program

Events

Meet with Apple

Apple Developer Centers