Protocol

INUIHostedViewControlling

Methods for presenting custom content in the Siri and Maps interfaces.

iOS 10.0+ | iPadOS 10.0+ | Mac Catalyst 13.1+ | visionOS 1.0+

```
@MainActor
protocol INUIHostedViewControlling : NSObjectProtocol
```

Overview

The `INUIHostedViewControlling` protocol defines methods for presenting a custom interface from Siri or Maps. You adopt this protocol in a view controller, which you then deliver in an Intents UI app extension as part of your iOS app. When displaying information to the user, Siri and Maps incorporate your view controller's content into the default interface. In iOS 11 and later, SiriKit gives you the option to replace some or all of the default interface.

An Intents UI app extension works in tandem with a corresponding Intents app extension. The Intents app extension handles the intent and provides a response that includes data from your app. The view controller in your Intents UI app extension receives that data in the form of an `INInteraction` object and uses it to configure its view. Using the methods of this protocol, you can replace some or all of the default Siri or Maps interface or you can augment the default interface with a custom view.

Each Intents UI app extension must contain only one view controller, but that view controller may provide a custom interface for multiple intents. Use the `IntentsSupported` key in the extension's `Info.plist` file to specify which intent classes the view controller supports. During the configuration phase, use the class of the intent to configure the contents of your view.

For more information on how to create an Intents UI extension, see Creating an Intents UI Extension.

Replace Some or All of the Default Interface

Prior to displaying an interface, SiriKit determines which pieces of data that it wants to display from the intent and your response and packages them into `INParameter` objects. For each parameter (or grouped set of parameters), SiriKit creates an instance of your view controller and calls its `configureView(for:of:interactiveBehavior:context:completion:)` method. Use that method to configure the custom interface that you want displayed by Siri and Maps. If you don't provide a custom view, SiriKit displays a default view.

> **Note**
>
> Before calling the `configureView(for:of:interactiveBehavior:context:completion:)` method with parameters, SiriKit calls it once with no parameters so that you can configure your interface all at once or provide an extra custom view.

Each view controller object may display parameters other than the ones that SiriKit provided. For example, if SiriKit asks you to display the pickup date for a ride, you could display the pickup date, drop-off date, and the pickup location in your custom view. When executing the handler block of the `configureView(for:of:interactiveBehavior:context:completion:)` method, return `INParameter` objects for all of the parameters that you displayed. SiriKit doesn't ask you to configure the same parameter twice.

Augment the Default Interface with a Single View

If you want to keep the default Siri or Maps interface and augment it with a custom view, implement the `configure(with:context:completion:)` method. When you implement only that method, SiriKit doesn't replace any of the default interfaces. Instead, it inserts your view controller's view into the default interface. You might augment the default interface when you want to take a more minimal approach to your customizations. For example, you might use it solely to add branding information to the default interface.

When augmenting the default interface, configure the contents of your view however you want. You can add subviews, embed child view controllers, and install layout constraints to display and manage your content. However, don't add gesture recognizers or views that involve user interactions to your view hierarchy. The system actively prevents the delivery of touch events to your view controller while it's onscreen. If you need to update your views dynamically, use a timer to trigger programmatic updates.

To avoid the duplication of some types of content, implement the properties of the `INUIHostedViewSiriProviding` protocol in your view controller. The properties define elements of the default interface that SiriKit can hide. For example, a ride booking app that provides its own map

might implement the `displaysMap` property and return true to cause SiriKit to hide the default map.

> **Important**
>
> If your view controller implements the `configureView(for:of:interactiveBehavior:context:completion:)` method, SiriKit doesn't call the `configure(with:context:completion:)` method.

Topics

Configuring the View Controller

```
func configureView(for: Set<INParameter>, of: INInteraction, interactiveBehavior: INUIInteractiveBehavior, context: INUIHostedViewContext, completion: (Bool, Set<INParameter>, CGSize) -> Void)
```

```
enum INUIInteractiveBehavior
```

Constants indicating how users are able to interact with your interface.

```
func configure(with: INInteraction, context: INUIHostedViewContext, completion: (CGSize) -> Void)
```

Configures your view controller's content.

Getting Information

```
enum INUIHostedViewContext
```

Constants indicating where your view controller appears.

Relationships

Inherits From

`NSObjectProtocol`

See Also

Custom UI for Siri and Maps

Creating an Intents UI Extension

Create an Intents UI app extension to customize the interfaces displayed by Siri and Maps.

`protocol` `INUIHostedViewSiriProviding`

Methods for hiding portions of the default interfaces that Siri provides.

`class` `INParameter`

A parameter of an interaction object.

`protocol` `INIntentSetImageKeyPath`