Article

Configuring the View Controller for Your Custom Interface

Configure your view controller to replace or augment the default interface in Siri or Maps.

Overview

When displaying responses to the user, SiriKit provides a default interface for displaying the content of your response. Replace some or all of this default interface with custom views, which you might do to fully customize the user's Siri-based interactions with your app. Another option is to augment the default interface with a single view, hiding portions of the default UI that your content might duplicate.

Requirements and Limitations

For all customizations, you provide a view controller whose views contain the information that you want to display. When replacing the default interface, SiriKit may create multiple instances of your view controller, and you must configure each one appropriately for the type of displayed content. When augmenting the default interface, SiriKit creates a single instance of your view controller.

While onscreen, your view controllers remain part of the foreground Siri or Maps interface until the user dismisses it. You can update your view controllers as needed using timers or other programmatic means. Your view controllers also receive the normal callbacks when they're loaded, shown, and hidden. However, your view controllers don't receive touch events or any other events while they're onscreen, and you can't add gesture recognizers to them. Therefore, never create an interface with controls or views that require user interactions.

Replace Some or All of the Default Interface

In iOS 11 and later, you can replace some or all of the default Siri or Maps interface. Before displaying an interface to the user, Siri builds a list of parameters—that is, instances of the `INParameter` class—representing the data it wants to display. Each parameter identifies a property of the intent or response object contained in the `INInteraction` object that SiriKit provides.

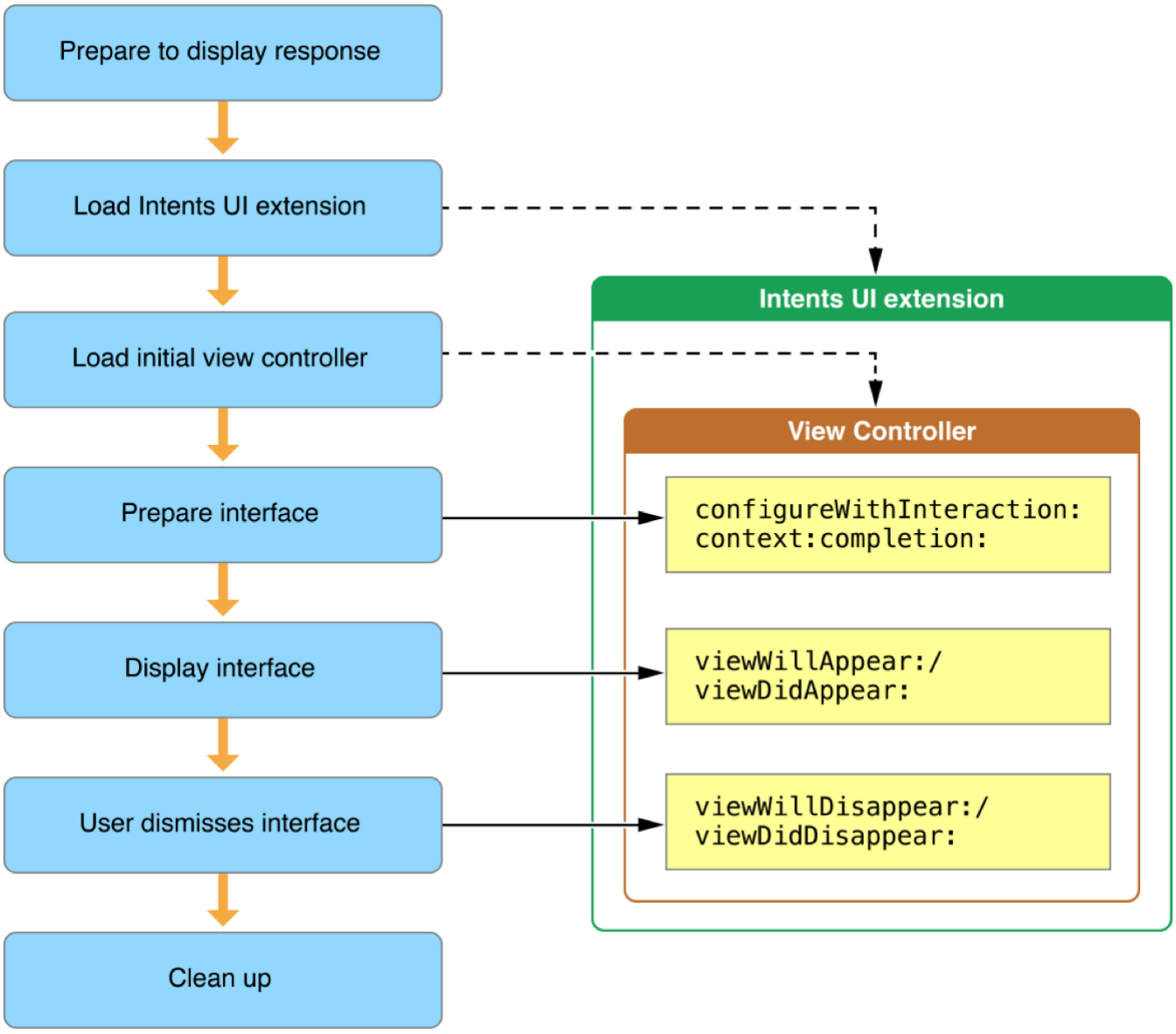
After building the list of parameters, SiriKit creates one or more instances of your view controller and calls the `configureView(for:of:interactiveBehavior:context:completion:)` method of each one, passing it a specific set of parameters. When the `configureView(for:of:interactiveBehavior:context:completion:)` method is called, you must decide whether you want to provide a custom interface for the specified parameters. If you do, add subviews to the view controller's root view and update those views with the parameter's content.

Although SiriKit often passes only one parameter at a time to your `configureView(for:of:interactiveBehavior:context:completion:)` method, you can configure a view that displays content for any number of parameters. When calling the completion handler of your configuration method, provide SiriKit with the set of parameters that your interface displays. Before creating a new view controller with the next set of parameters, SiriKit checks to see if you already displayed those parameters. If you did, SiriKit does not ask you to display them again.

Augment the Default Interface

If you want to keep the default Siri or Maps interface and only augment it with your custom content, implement the `configure(with:context:completion:)` method of your view controller. SiriKit instantiates your view controller and calls that method to configure your view controller's view. The context parameter tells you where your view controller will be displayed, giving you an opportunity to display your content differently in Siri and Maps.

The figure below shows the high-level life cycle of your view controller when you configure it to augment the default interface. The system creates your view controller and calls its `configure(with:context:completion:)` method, passing it the interaction object you need to configure your interface. Once configured, your view controller is presented onscreen with the rest of the Siri or Maps content.



> **Note**
>
> While a view controller is onscreen, Maps may call the `configure(with:context:completion:)` method again to deliver updated information for you to display. For example, Maps calls this method when your Intents extension provides a status update for a booked ride. Use any follow-up calls to update your view controller's interface.

When the user dismisses the Siri or Maps interface, the system releases its reference to your view controller and your Intents UI extension. Your view controllers should only display information. Do not try to save data or communicate with your app when your view controller moves offscreen.

Important

In iOS 11 and later, if your view controller implements both the `configureView(for:of:interactiveBehavior:context:completion:)` and `configure(with:context:completion:)` methods, SiriKit calls only the `configureView(for:of:interactiveBehavior:context:completion:)` method.

Tips for Implementing Your View Controller

Here are some tips for implementing the view controller for your Intents UI extension:

- **Incorporate your brand into your interface.** Using your app's color, imagery, and other design elements is a great way to add familiarity and convey the presence of your app within the Siri or Maps interfaces.
- **Configure any animated content to run only when your view controller is visible.** Wait until your view controller's `viewDidAppear(_:)` method is called to start animations. Stop animations in your view controller's `viewWillDisappear(_:)` method.
- **Configure your view controller's view as quickly as possible so that Siri can display it.** Your view controller may not be onscreen for very long, so use local resources and the provided `INInteraction` object for the bulk of your configuration. If you need to fetch more information from a server, always do so asynchronously and update your interface later.
- **Configure your interface using only the provided interaction object.** The provided `INInteraction` object contains the original intent and the response provided by your Intents extension, which runs in a separate process. Using only the provided interaction object ensures that your view controller's interface accurately reflects the information provided by your Intents extension. To communicate more information to your Intents UI app extension, your Intents app extension can include a custom `NSUserActivity` object with its response and put the additional information in that object's `userInfo` dictionary.
- **Do not include advertising in your interface.** You may include branding and information that is relevant to the user, but advertising is prohibited.
- **Return a zero size when you want to hide your custom interface.** When calling the handler block of your configuration method, specify `CGRectZero` for the size when you do not want the view controller to be shown onscreen. You might hide the view controller when there is no additional information to display for the specified intent.
- **Do not include a map view when your view controller is shown in a Maps context.** When the context parameter is set to `INUIHostedViewContext.mapsCard`, do not include an `MKMapView` in your view controller's interface. Maps already displays a map, so having two maps showing similar information would be confusing to users.

- **Use child view controllers to switch between different types of content.** Your Intents UI app extension has only one initial view controller. Using child view controllers lets you encapsulate the behavior for for different intents or different views of an intent in one place. During configuration of your initial view controller, all you have to do is instantiate the appropriate child view controller and install its view in your interface.

Hiding Portions of the Default Interface

SiriKit lets you hide some types of content in the default interfaces using properties of the `INUIHostedViewSiriProviding` protocol. If you are only augmenting the default interface with custom content, use these properties to avoid duplicating the content shown by Siri or Maps. For example, a ride booking app that adds a map to the default interface could implement the `displaysMap` property and use it to hide the map in the default interface.

For more information about hiding portions of the default interface, see `INUIHostedViewSiriProviding`.

See Also

Articles

Adding User Interactivity with Siri Shortcuts and the Shortcuts App

Add custom intents and parameters to help users interact more quickly and effectively with Siri and the Shortcuts app.

Defining Relevant Shortcuts for the Siri Watch Face

Inform Siri when your app's shortcuts may be useful to the user.

Deleting Donated Shortcuts

Remove your donations from Siri.

Dispatching intents to handlers

Provide SiriKit with an intent handler capable of handling a specific intent.

Improving Siri Media Interactions and App Selection

Fine-tune voice controls and improve Siri Suggestions by sharing app capabilities, customized names, and listening habits with the system.

Improving interactions between Siri and your messaging app

Donate app-specific content, use Siri's contact suggestions, and adopt the latest platform features to create a more consistent messaging experience.

- Registering Custom Vocabulary with SiriKit

  Register your app's custom terminology, and provide sample phrases for how to use your app with Siri.

- Confirming the Details of an Intent

  Perform final validation of the intent parameters and verify that your services are ready to fulfill the intent.

- Handling an Intent

  Fulfill the intent and provide feedback to SiriKit about what you did.

- Resolving the Parameters of an Intent

  Validate the parameters of an intent and make sure that you have the information you need to continue.

- Generating a List of Ride Options

  Generate ride options for Maps to display to the user.

- Handling the Ride-Booking Intents

  Support the different intent-handling sequences for booking rides with Shortcuts or Maps.

- Donating Reservations

  Inform Siri of reservations made from your app.

- Specifying Synonyms for Your App Name

  Provide alternative names for your app that are more familiar or easier for users to speak.

- Intent Phrases

  The keys that you include in your global vocabulary file to show how users engage your app from Siri.