CarPlay App Programming Guide

March 2024

Developer

Table of Contents

Introduction

CarPlay is a smarter, safer way to use your iPhone in the car. CarPlay takes the things you want to do with your iPhone while driving and puts them right on your car's built-in display.

In addition to getting directions, making calls, sending and receiving messages, and listening to music, CarPlay supports the following types of apps.

- Audio
- Communication (messaging and calling)
- Driving task
- EV charging
- Fueling
- Navigation (turn-by-turn directions)
- Parking
- Quick food ordering

This guide describes how to create these types of CarPlay apps.

Note This guide does not cover CarPlay automaker apps (published by automakers).

Overview

People download CarPlay apps from the App Store and use them on iPhone like any other app. When connected to a CarPlay vehicle, the app icon appears on the CarPlay home screen. CarPlay apps are not separate apps—you add CarPlay support to your existing app.

CarPlay apps are designed to look and feel like your app on iPhone, but with UI elements that are similar to built-in CarPlay apps.

Your app uses the CarPlay framework to present UI elements to the user. iOS manages the display of UI elements and handles the interface with the car. Your app does not need to manage the layout of UI elements for different screen resolutions, or support different input hardware such as touchscreens, knobs, or touch pads.

CarPlay apps must meet the basic requirements defined in the *CarPlay Entitlement Addendum*, and must follow the CarPlay App Guidelines.

For general design guidance, see Human Interface Guidelines for CarPlay.

CarPlay app entitlements

All CarPlay apps require a CarPlay app entitlement that matches your app type.

To request a CarPlay app entitlement, go to http://developer.apple.com/carplay and provide information about your app, including the type of entitlement that you are requesting. You also need to agree to the CarPlay Entitlement Addendum.

Apple will review your request. If your app meets the criteria for a CarPlay app, Apple will assign a CarPlay app entitlement to your Apple Developer account and notify you.

CarPlay app guidelines

All CarPlay apps must adhere to the following guidelines.

Guidelines for all CarPlay apps

1. Your CarPlay app must be designed primarily to provide the specified feature to a user (e.g. CarPlay audio apps must be designed primarily to provide audio playback services, CarPlay parking apps must be designed primarily to provide parking services, etc.).
2. Never instruct users to pick up their iPhone to perform a task. If there is an error condition, such as a required log in, you can let users know about the condition so they can take action when safe. However, user messages must not include wording that asks users to manipulate their iPhone.
3. All CarPlay user flows must be possible without interacting with iPhone.
4. All CarPlay user flows must be meaningful to use while driving. Don't include features in CarPlay that aren't related to the primary task (e.g. unrelated settings, maintenance features, etc.).
5. No gaming or social networking.
6. Never show the content of messages, texts, or emails on the CarPlay screen.
7. Use templates for their intended purpose, and only populate templates with the specified information types (e.g. a list template must be used to present a list for selection, album artwork in the now playing screen must be used to show an album cover, etc.).
8. All voice interaction must be handled using SiriKit (with the exception of CarPlay navigation apps, see below).

Additional guidelines for CarPlay audio apps

1. Never show song lyrics on the CarPlay screen.

Additional guidelines for CarPlay communication (messaging and calling) apps

1. Communication apps must provide either short form text messaging features, VoIP calling features, or both.
2. Email is not considered short form text messaging and is not permitted.
3. Communication apps that provide text messaging features must support all 3 of the following SiriKit intents:
    - Send a message (`INSendMessageIntent`)
    - Request a list of messages (`INSearchForMessagesIntent`)
    - Modify the attributes of a message (`INSetMessageAttributeIntent`)
4. Communication apps that provide VoIP calling features must support CallKit, and all of the following SiriKit intents:
    - Start a call (`INStartCallIntent`)
    - Start an audio-only call (`INStartAudioCallIntent`) required for apps that support iOS 14 and earlier
    - Request a list of calls (`INSearchCallHistoryIntent`) required for apps that support iOS 14 and earlier

Additional guidelines for CarPlay driving task apps

1. Driving task apps must enable tasks people *need* to do while driving. Tasks must actually *help* with the drive, not just be tasks that are done while driving.
2. Driving task apps must use the provided templates to display information and provide controls. Other kinds of CarPlay UI (e.g. custom maps, real-time video) are not possible.
3. Do not show CarPlay UI for tasks unrelated to driving (e.g. account setup, detailed settings).
4. Do not periodically refresh data items in the CarPlay UI more than once every 10 seconds (e.g. no real-time engine data).
5. Do not periodically refresh points of interest in the POI template more than once every 60 seconds.
6. Do not create POI (point of interest) apps that are focused on finding locations on a map. Driving tasks apps must be primarily designed to accomplish tasks and are not intended to be location finders (e.g. store finders).
7. Use cases outside of the vehicle environment are not permitted.

Additional guidelines for CarPlay EV charging apps

1. EV charging apps must provide meaningful functionality relevant to driving (e.g. your app can't just be a list of EV chargers).
2. When showing locations on a map, do not expose locations other than EV chargers.

Additional guidelines for CarPlay fueling apps

1. Fueling apps must provide meaningful functionality relevant to driving (e.g. your app can't just be a list of fueling stations).
2. When showing locations on a map, do not expose locations other than fueling stations.

Additional guidelines for CarPlay parking apps

1. Parking apps must provide meaningful functionality relevant to driving (e.g. your app can't just be a list of parking locations).
2. When showing locations on a map, do not expose locations other than parking.

Additional guidelines for CarPlay navigation (turn-by-turn directions) apps

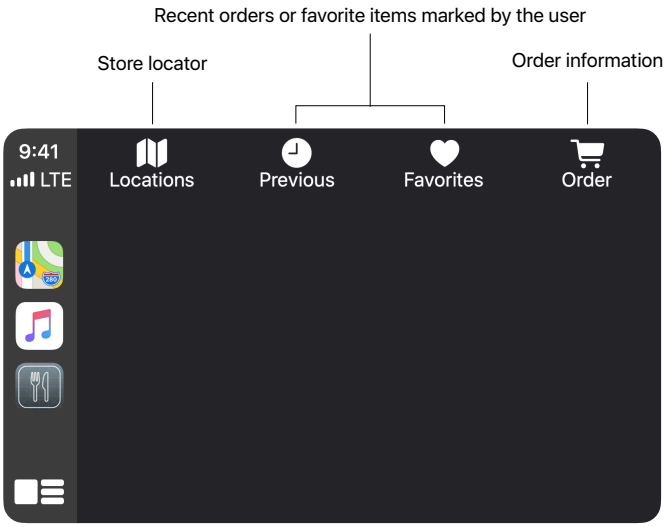
1. Navigation apps must provide turn-by-turn directions with upcoming maneuvers.
2. The base view must be used exclusively to draw a map. Do not draw windows, alerts, panels, overlays, or user interface elements in the base view. For example, don't draw lane guidance information in the base view. Instead, draw lane guidance information as a secondary maneuver using the provided template.
3. Use each provided template for its intended purpose. For example, maneuver images must represent a maneuver and cannot represent other content or user interface elements.
4. Provide a way to enter panning mode. If your app supports panning, you must include a button in the map template that allows the user to enter panning mode since drag gestures are not available in all vehicles. Drag gestures must only be used for panning the map.
5. Immediately terminate route guidance when requested. For example, if the user starts route guidance using the vehicle's built-in navigation system, your app delegate will receive a cancelation notification and must immediately stop route guidance.
6. Correctly handle audio. Voice prompts must work concurrently with the vehicle's audio system (such as the user listening to the car's FM radio) and your app should not needlessly activate audio sessions when there is no audio to play.
7. Ensure that your map is appropriate in each supported country.
8. Be open and responsive to feedback. Apple may contact you in the event that Apple or automakers have input to design or functionality.
9. Voice control must be limited to navigation features.

Additional guidelines for quick food ordering apps

1. Quick food ordering apps must be Quick Service Restaurant (QSR) apps designed primarily for driving-oriented food orders (e.g. drive thru, pick up) when in CarPlay and are not intended to be general retail apps (e.g. supermarkets, curbside pickup).
2. Quick food ordering apps must provide meaningful functionality relevant to driving (e.g. your app can't just be a list of store locations).
3. Simplified ordering only. Don't show a full menu. You can show a list of recent orders, or favorites limited to 12 items each.
4. When showing locations on a map, do not expose locations other than your Quick Service Restaurants.

   The following example shows how to structure a quick food ordering app in CarPlay. The app provides four tabs which allow the user select a store, view a list of recent orders or favorite items, and confirm order information. The icons and text may be customized.

   Locations, lists, and information screens are limited to 12 items. Quick food ordering user flows should be simple and limited to the most common tasks. Show only the most important and relevant information.



Tab bar in a quick food ordering app

Development environment

Xcode and an Apple Developer Program account are required to create apps for CarPlay.

Entitlements

Once you have received a CarPlay app entitlement, create a new Provisioning Profile that includes the CarPlay app capability.

1. Log in to your Apple Developer Account https://developer.apple.com/account/.
2. Under **Certificates**, **IDs & Profiles**, select **Identifiers**.
3. Select the App ID associated with your app, or create a new App ID.
4. Select the **Additional Capabilities** tab.
5. Enable all necessary CarPlay app entitlements for your app.
6. Click Save on the top right.
7. Continue to **Provisioning Profiles** and create a new provisioning profile for your App ID.

For additional information, see Developer Account Help.

https://developer.apple.com/help/account/

After you have created a new Provisioning Profile, import it into Xcode. Xcode and Simulator require a Provisioning Profile that supports CarPlay.

In Xcode, create an `Entitlements.plist` file in your project, if you don't have one already. Add your CarPlay app entitlement keys as a boolean key. The following example is for a CarPlay audio app.

```
<key>com.apple.developer.carplay-audio</key>
<true/>
```

In Xcode, under *Signing & Capabilities* turn off *Automatically manage signing*, and under *Build Settings* ensure that *Code Signing Entitlements* is set to the path of your `Entitlements.plist` file.

Once a CarPlay app entitlement is added to your app, your app icon will appear on the CarPlay home screen. You cannot selectively show or hide CarPlay for certain users. Only publish your app with CarPlay support when you are ready for everyone to see it.

See Sample Code for project examples.

Use the entitlement key(s) that match your selected provisioning profile.

Entitlement	Key	Minimum iOS version
CarPlay Audio App (CarPlay framework) App supports the CarPlay framework. If your app supports iOS 13 or earlier, this entitlement may be combined with the deprecated CarPlay Audio App (Media Player framework) entitlement.	`com.apple.developer.carplay-audio`	iOS 14
CarPlay Communication App App supports the CarPlay framework, and SiriKit intents for messaging or VoIP calling apps. If your app supports iOS 13 or earlier, this entitlement may be combined with the deprecated CarPlay Messaging App entitlement and/or CarPlay VoIP Calling App entitlement.	`com.apple.developer.carplay-communication`	iOS 14
CarPlay Driving Task App	`com.apple.developer.carplay-driving-task`	iOS 16
CarPlay EV Charging App This entitlement may be combined with the CarPlay Fueling App entitlement.	`com.apple.developer.carplay-charging`	iOS 14
CarPlay Fueling App This entitlement may be combined with the CarPlay EV Charging App entitlement.	`com.apple.developer.carplay-fueling`	iOS 16
CarPlay Navigation App	`com.apple.developer.carplay-maps`	iOS 12
CarPlay Parking App	`com.apple.developer.carplay-parking`	iOS 14
CarPlay Quick Food Ordering App	`com.apple.developer.carplay-quick-ordering`	iOS 14

Deprecated entitlements

Audio apps can support the CarPlay framework (recommended), Media Player framework (deprecated), or both. Be sure to include the correct entitlement(s) to match the framework(s) your app actually supports. On iOS 14 and later, the CarPlay framework will be used if your app supports both frameworks.

If your app needs to work on iOS 13 and earlier, also support the Media Player framework and include the `com.apple.developer.playable-content` entitlement. Apps that only support the Media Player framework will work on later versions of iOS, but without a customized user interface.

Entitlement	Key
CarPlay Audio App (Media Player framework) **Deprecated.** App supports the Media Player framework. Include both CarPlay audio app entitlements if your app supports the CarPlay framework and the Media Player framework.	`com.apple.developer.playable-content`

Communication apps can support the CarPlay framework in addition to SiriKit and CallKit. Be sure to include the correct entitlement(s) to match the frameworks and features you support.

If your app needs to work on iOS 13 and earlier, also include the `com.apple.developer.carplay-messaging` and/or `com.apple.developer.carplay-calling` entitlements to match your app features. Apps that don't support the CarPlay framework will still work on later versions of iOS, but without a customized user interface.

All communication apps must support required SiriKit intents, and CallKit (for calling apps). For a list of required SiriKit intents for communication apps, see CarPlay app guidelines.

Entitlement	Key
CarPlay Messaging App **Deprecated.** App relies solely on SiriKit and supports SiriKit intents to send, request, and modify messages. May be combined with the CarPlay Communication App entitlement, and the optional CarPlay VoIP Calling App entitlement to support iOS 13 and earlier.	`com.apple.developer.carplay-messaging`
CarPlay VoIP Calling App **Deprecated.** App relies solely on SiriKit and CallKit, and supports SiriKit intents for starting calls and requesting a list of calls. May be combined with the CarPlay Communication App entitlement, and the optional CarPlay Messaging App entitlement to support iOS 13 and earlier.	`com.apple.developer.carplay-calling`

Simulators

Apple provides two simulators to help you develop and test your CarPlay app. CarPlay Simulator is a tool that simulates a complete car environment and requires you to install your app on iPhone. Xcode Simulator includes a CarPlay window that lets you run and debug your CarPlay UI. It's recommended that you download and use CarPlay Simulator to closely match the behavior of CarPlay in a car.

CarPlay Simulator

CarPlay Simulator is a standalone Mac app that simulates a complete car environment. CarPlay Simulator is included in the **Additional Tools for Xcode** package which you can download from https://developer.apple.com/download/all/.

Locate **CarPlay Simulator** in the **Hardware** folder, run it, and connect iPhone using a USB cable. CarPlay starts on iPhone just the same as if you had it connected to a real car.

Xcode Simulator

Xcode Simulator lets you run and debug your CarPlay UI in a second window. The window acts as the car's display and allows you to interact with your CarPlay app in a similar manner to when you are connected to a CarPlay system.

To access CarPlay in Xcode Simulator, launch Simulator and select **I/O**, **External Displays**, and **CarPlay** to show a CarPlay screen.

Xcode Simulator is useful for regular build and test cycles for your CarPlay UI, but you should not rely exclusively on Xcode Simulator for all CarPlay app development. Here are some scenarios that require CarPlay Simulator or an actual CarPlay environment, and cannot be tested using Xcode Simulator.

- Testing while iPhone is locked. Most users interact with CarPlay while iPhone is locked so you need to ensure that your app works correctly even when iPhone is locked.
- Testing runtime scenarios such as switching between CarPlay and the car's built-in UI, or connecting and disconnecting iPhone.
- Testing scenarios where the car is playing audio. Remember that additional audio sources may be playing while CarPlay is active and your app must be a good audio citizen. For example, activating an audio session in your app has the side effect of immediately stopping the car's FM radio so you must only activate your audio session when you are ready to play audio.
- Testing Siri features with your app.
- Testing features that depend on location.
- Testing your navigation app with instrument cluster displays.

Testing using a vehicle or aftermarket head unit

You can also test your CarPlay app using an actual vehicle or an aftermarket head unit with a power supply. If you use an aftermarket head unit, choose one that supports wireless CarPlay so you can simultaneously connect iPhone to the head unit and to Xcode on your Mac using a cable.

Templates

CarPlay apps are built from a fixed set of UI templates that iOS renders on the CarPlay screen.

CarPlay apps are responsible for selecting which template to show on the screen (the controller), and providing data to be shown inside the template (the model). iOS is responsible for rendering the information in CarPlay (the view).

The CarPlay framework includes general purpose templates such as alerts, lists, and tab bars. It also includes templates designed for specific tasks such as contacts, maps, and now playing.

Each CarPlay app type supports specific templates and this is governed by the app entitlement. Attempting to use an unsupported template triggers an exception at runtime.

	Audio	Communication	Navigation	Driving task, EV charging, fueling, parking, and quick food ordering
Action Sheet	● [*1]	●	●	●
Alert	●	●	●	●
Grid	●	●	●	●
List	●	●	●	●
Tab bar	●	●	●	●
Information		●	●	●
Point of Interest				●
Now Playing	●	● [*1]		
Contact		●	●	
Map			●	
Search			●	
Voice control			●	

[*1] New in iOS 17.

There is a limit to the number of templates (depth of hierarchy) that you can push onto the screen. Most apps are limited to a depth of 5 templates. Fueling apps are further limited to 3 templates, and driving task and quick food ordering apps are limited to 2 templates. These include the root template.

Action sheet

An action sheet is a specific style of alert that appears in response to a control or action, and presents a set of two or more choices related to the current context. Use action sheets to let people initiate tasks, or to request confirmation before performing a potentially destructive operation.

Action sheet

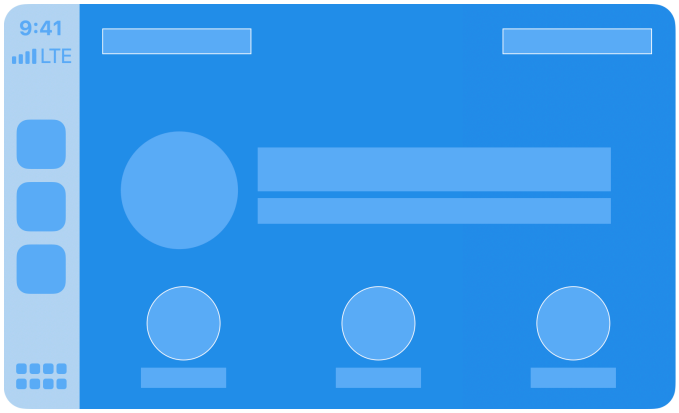
Alert

Alerts convey important information related to the state of your app. An alert consists of a title and one or more buttons. You can provide titles of varying lengths and let CarPlay choose the title that best fits the available screen space. If underlying conditions permit, alerts can be dismissed programatically.

Alert

Contact

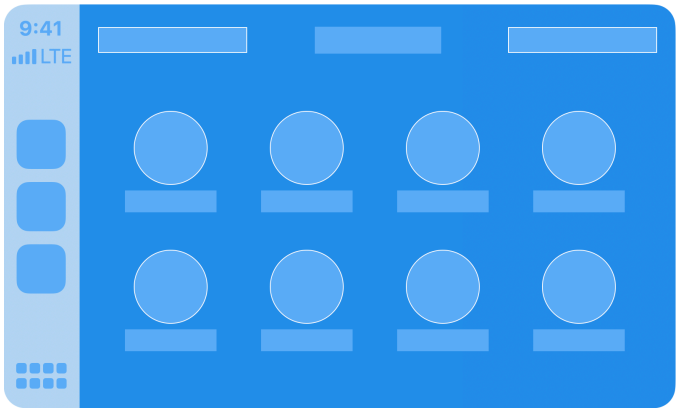
Contacts allow you to present information about a person or business. A contact consists of an image, title, subtitle, and action buttons. Use action buttons to let users perform tasks related to the current contact, such as making a phone call or sending a message.



Contact

Grid

A grid is a specific style of menu that presents up to eight choices represented by an icon and a title. Use the grid template to let people select from a fixed list of items. The grid also includes a navigation bar with a title, leading buttons, and trailing buttons which can be shown as icons or text.



Grid

Information

An information screen is a specific style of list that presents a limited number of static labels with optional footer buttons. Labels can appear in a single column or in two columns. Starting in iOS 16, the information template can also include leading and trailing navigation bar buttons.

Use the information template to show important information. For example, an EV charging app may display information about a charging station such as availability, while a quick food ordering app may display an order summary such as pick-up location and time.

Since the number of labels is limited, show only the most important summary information needed to complete a task.

Information

List

A list presents data as a scrolling, single-column table of rows that can be divided into sections. Lists are ideal for text-based content, and can be used as a means of navigation for hierarchical information.

Each item in a list can include attributes such as an icon, title, subtitle, disclosure indicator, progress indicator, playback status, or read status. Use a general list item if you just need to show an icon with text, or choose a specific list item such as the image row list item which is useful in audio apps, or the messages list item which is useful in communication apps.

Some cars dynamically limit lists to 12 items. You can check for the maximum number of items, but you always need to be prepared to handle the case where only 12 items are shown.

If your app supports SiriKit, you can add an “Ask Siri ...” item that appears in the list.

List

List with an image row list item

Now playing

The now playing screen presents information about the currently playing audio, such as title, artist, elapsed time, and album artwork. It also lets people control your app using playback control buttons.

The now playing screen is customizable and you should adapt it to your needs. For example, you can provide a link to upcoming tracks, the playback control buttons can be customized with your own icons, and the elapsed time indicator can be configured for fixed-length audio or for open ended audio such as a live stream.

The now playing template is special because users can directly access it from the CarPlay home screen or through the now playing button in your app's navigation bar. You must be prepared to populate the now playing template at all times.

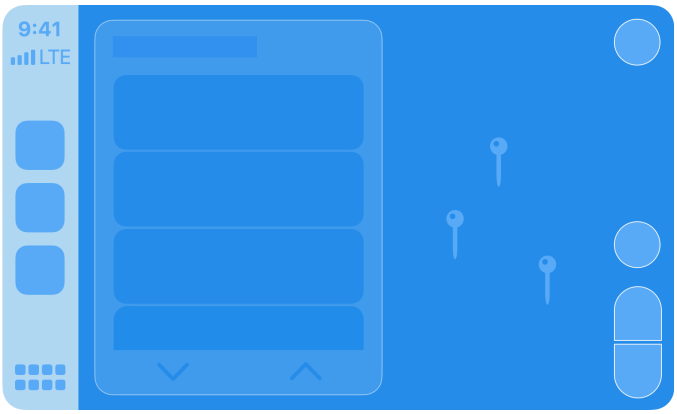
Only the list template may be pushed on top of the now playing template. For example, if your app enables the "Playing Next" button in the now playing template, you can respond by showing a list template containing the upcoming playback queue.

Now playing

Point of interest

A point of interest screen lets the user browse nearby locations on a map and choose one for further action.

The point of interest template includes a map provided by the MapKit framework, and an overlay containing a list of up to 12 locations with customizable pin images. Starting in iOS 16, you may optionally provide a larger pin image for the currently selected location. The list of locations should be limited to those that are most relevant or nearby.



Point of interest

Tab bar

The tab bar is a versatile container for other templates, where each template occupies one tab in the tab bar. People can use the tab bar to rapidly switch between different templates.

Use `CPTabBarTemplate maximumTabCount` to determine the maximum number of tabs that can be displayed. In current versions of iOS, the tab bar allows up to 4 tabs for audio apps and up to 5 tabs for all other app types, although this may change in the future.

When your app is playing audio, CarPlay displays a now playing button in the top right corner of the tab bar for easy access to playback controls. The now playing button may not appear if your tab bar has more than 4 tabs.

Tab bar

Notifications

Notifications are supported in CarPlay communication, EV Charging, and parking apps.

Notifications should be used sparingly in CarPlay and must be reserved for important tasks required while driving. Do not use notifications in CarPlay for features that are only relevant when using your app on iPhone. In general, notifications are not read aloud in CarPlay.

Note that route guidance notifications in CarPlay navigation apps are handled by the CarPlay framework itself and are not part of the standard app notification mechanism.

Request authorization to show notifications

In order to show notifications in CarPlay, include the `carPlay` option when requesting authorization for notifications.

Users can use Settings to show or hide your app's notifications in CarPlay. Gracefully disable notification-related features if the user declines to show notifications in CarPlay.

```
let authorizationOptions : UNAuthorizationOptions = [.badge, .sound, .alert, .carPlay]

let notificationCenter = UNUserNotificationCenter.current()
notificationCenter.requestAuthorization(options: authorizationOptions) {
    (granted, error) in
    // Enable or disable app features based on authorization
}
```

Create a notification category with the CarPlay option

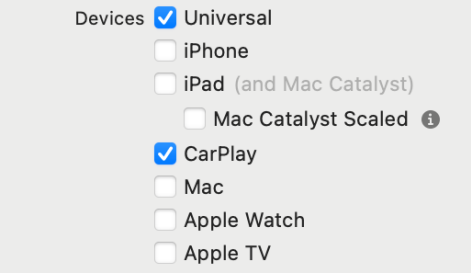
In addition to requesting authorization, your app must enable CarPlay for the notification categories you want displayed. To enable CarPlay, create a notification category with the `allowInCarPlay` option. Assign an identifier to the category, and make sure that any local or remote notifications for messages have the same category identifier.

If you are developing a CarPlay communication app, also see Implementing communication notifications for more details on messaging notifications. In CarPlay, notifications must only include information such as the sender and group name in the title and subtitle. The contents of the message must never be shown in CarPlay.

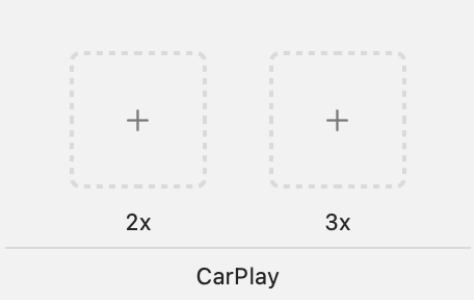
Assets

Prepare CarPlay assets for images used in templates such as icons and buttons. Note that CarPlay supports multiple scales and both light and dark interfaces so you should take this into account when creating assets. Create versions that are suitable for 2x and 3x scale factors, and for light and dark styles.

Turn on CarPlay assets in Xcode and populate the CarPlay 2x and 3x image wells.



Turn on CarPlay assets



Populate the CarPlay 2x and 3x image wells

Use the following size guidance when creating images.

	Maximum size in points	Maximum size in pixels (3x)	Maximum size in pixels (2x)
Contact action button	50pt x 50pt	150px x 150px	100px x 100px
Grid icon	40pt x 40pt	120px x 120px	80px x 80px
Now playing action button	20pt x 20pt	60px x 60px	40px x 40px
Tab bar icon	24pt x 24pt	72px x 72px	48px x 48px

If you create assets programmatically, use `UIImageAsset` to combine `UIImage` instances into single image with both light and dark trait collections.

If you need to know the CarPlay screen scale at runtime, use the trait collection `carTraitCollection` to obtain the display scale. Don't use other parameters in the `carTraitCollection` and be sure to get the scale for the car's screen (not the scale for the iPhone screen).

To determine the sizes of images used in lists, use `maximumImageSize` in `CPListItem` and `CPListImageRowItem` to obtain the maximum image size and provide images with matching resolution.

Use CarPlay Simulator to test your app and see how it appears under different conditions, including screen resolutions, scale factors, and light/dark styles.

Audio handling

Playback

If your app plays audio, ensure that it works well with audio sources in the car.

Only activate your audio session the moment you are ready to play audio. When you activate your audio session, other audio sources in the car will stop. For example, if a user is listening to the car's FM radio and you active your audio session too soon, the FM radio will stop. People expect FM radio to continue to play until they explicitly choose to play an audio stream in your app. Don't simply activate your audio session at the time your app launches. Instead, wait until you actually need to play audio.

If you are developing a CarPlay navigation app, see Voice prompts for details on playing voice prompts for upcoming route maneuvers.

Recording

In general, recording is not supported while in CarPlay. If your app has recording features, don't enable them when CarPlay is active. If you activate an audio session with recording enabled, it can affect audio playback from other sources and impact audio input for the car's own functions such as voice assistants and phone calls. While in CarPlay, configure audio sessions without recording features.

An exception is for CarPlay navigation apps which use recording features for voice input. In CarPlay navigation apps, recording features may be used, but only in conjunction with the voice control template.

Build a CarPlay app

Startup

All CarPlay apps must adopt scenes and declare a CarPlay scene to use the CarPlay framework. You can declare a scene dynamically, or you can include an application scene manifest in your `Info.plist` file. The following is an example of an application scene manifest that declares a CarPlay scene. You can add this to the top level of your app's `Info.plist` file.

```
<key>UIApplicationSceneManifest</key>
<dict>
   <key>UISceneConfigurations</key>
    <dict>
        <!-- Declare device scene -->
        <key>UIWindowSceneSessionRoleApplication</key>
        <array>
            <dict>
                <key>UISceneClassName</key>
                <string>UIWindowScene</string>
                <key>UISceneConfigurationName</key>
                <string>Phone</string>
                <key>UISceneDelegateClassName</key>
                <string>MyAppWindowSceneDelegate</string>
            </dict>
        </array>
        <!-- Declare CarPlay scene -->
        <key>CPTemplateApplicationSceneSessionRoleApplication</key>
        <array>
            <dict>
                <key>UISceneClassName</key>
                <string>CPTemplateApplicationScene</string>
                <key>UISceneConfigurationName</key>
                <string>MyApp-Car</string>
                <key>UISceneDelegateClassName</key>
                <string>MyApp.CarPlaySceneDelegate</string>
            </dict>
        </array>
    </dict>
</dict>
```

In the above example, the app declares 2 scenes—one for the iPhone screen, and one for the CarPlay screen.

The name of the class that serves as the scene delegate is defined in the manifest by `UISceneDelegateClassName`. Your delegate must conform to `CPTemplateApplicationSceneDelegate`. Listen for the `didConnect` and `didDisconnect` methods to know when your app has been launched on the CarPlay screen. Remember, your app may be launched only on the CarPlay screen.

When your app is launched, you will receive a `CPInterfaceController` that manages all the templates on the CarPlay screen. Hold onto the controller since you'll need it to manage templates, such as showing a now playing screen or an alert.

On launch, you must also specify a root template. In the example below, the app specifies a CPListTemplate as the root template.

```
import CarPlay

class CarPlaySceneDelegate: UIResponder, CPTemplateApplicationSceneDelegate {
    var interfaceController: CPInterfaceController?

    // CarPlay connected

    func templateApplicationScene(_ templateApplicationScene: CPTemplateApplicationScene,
                                  didConnect interfaceController: CPInterfaceController) {
        self.interfaceController = interfaceController
        let listTemplate: CPListTemplate = ...
        interfaceController.setRootTemplate(listTemplate, animated: true)
    }

    // CarPlay disconnected

    func templateApplicationScene(_ templateApplicationScene: CPTemplateApplicationScene,
                                  didDisconnect interfaceController: CPInterfaceController) {
        self.interfaceController = nil
    }
}
```

Create a list template

The following example shows how to create a list containing a single list item with a title and a subtitle.

When the user selects a list item, your list item handler will be called. You should take appropriate action here, such as starting audio playback in the case of an audio app. If you initiate asynchronous work and don't immediately call the completion block, CarPlay will display a spinner to let the user know that your app is busy. When you're ready to continue, you must call the completion block to tell CarPlay to remove the spinner.

```
import CarPlay

let item = CPListItem(text: “My title", detailText: “My subtitle")
item.listItemHandler = { item, completion, [weak self] in

    // Start playback asynchronously…

    self.interfaceController.pushTemplate(CPNowPlayingTemplate.shared(), animated: true)
    completion()
}

let section = CPListSection(items: [item])
let listTemplate = CPListTemplate(title: "Albums", sections: [section])
self.interfaceController.pushTemplate(listTemplate, animated: true)
```

Create a now playing template

The now playing template is a shared instance so you need to obtain it and configure its properties.

Do this when the interface controller connects to your app because iOS can display the shared now playing template on your behalf. For example, when the user taps the “Now Playing” button on the CarPlay home screen or in your app’s navigation bar, iOS will immediately present the shared now playing template.

This example shows an app configuring the playback rate button on the now playing template.

```
import CarPlay

class CarPlaySceneDelegate: UIResponder, CPTemplateApplicationSceneDelegate {

    func templateApplicationScene(_ templateApplicationScene: CPTemplateApplicationScene,
                                  didConnect interfaceController: CPInterfaceController) {

        let nowPlayingTemplate = CPNowPlayingTemplate.shared()

        let rateButton = CPNowPlayingPlaybackRateButton() {
            // Change the playback rate!
        }
        nowPlayingTemplate.updateNowPlayingButtons([rateButton])
    }
}
```

Work while iPhone is locked

CarPlay is frequently used while iPhone is in a locked state. Test your app throughly to ensure it works as expected when iPhone is locked.

You won't be able to access any of the following when launched or running while iPhone is locked.

- Files saved with `NSFileProtectionComplete` or `NSFileProtectionCompleteUnlessOpen`.
- Keychain items with a `kSecAttrAccessible` attribute of `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly`, `kSecAttrAccessibleWhenUnlocked` or `kSecAttrAccessibleWhenUnlockedThisDeviceOnly`.

Launch other apps

If your app launches other apps in CarPlay, such as to get directions or make a phone call, use the `CPTemplateApplicationScene open(_:options:completionHandler:)` method to launch the other app using a URL to ensure it launches on the CarPlay screen.

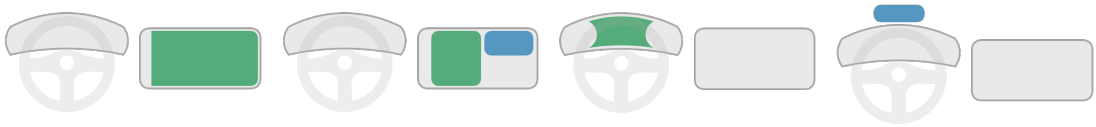
Build a CarPlay navigation app

The following section describes how to create a CarPlay navigation app.

CarPlay navigation apps have additional UI elements and capabilities that are different from other CarPlay app types. Skip this section if you are not creating a navigation app.

Supported displays

CarPlay navigation apps can appear in the center display, the CarPlay Dashboard, and the instrument cluster in supported vehicles. In addition, CarPlay navigation apps can supply metadata for vehicles that display information in the instrument cluster or HUD (head-up display) in a wide variety of vehicles. Support all capabilities in your app for a seamless experience in all vehicle configurations.



	Map in center display	Map in CarPlay dashboard	Map in instrument cluster	Metadata in instrument cluster or HUD
iOS 12	●			
iOS 13.4	●	●		
iOS 16.4	●	●	●	
iOS 17.4	●	●	●	●

Additional templates for navigation apps

CarPlay navigation apps use additional templates to display map information, a keyboard, and voice control feedback.

Base View

All CarPlay navigation apps start with a base view. The base view is where you draw your map. Create the base view and attach it to the provided window when CarPlay starts.

The base view must be used exclusively to draw a map, and cannot be used to draw alerts, overlays, or other UI elements. All UI elements that appear on the screen, including the navigation bar and map buttons, must be implemented using other templates. Your app won't receive direct tap or drag events in the base view.

You will be required to draw your map on a variety of screens with different aspect ratios, resolutions, and in light or dark mode. Get the current mode using `contentStyle` in your CarPlay template application scene and receive `contentStyleDidChange` notifications in your scene delegate. You must also consider the safe area (the portion of the map not obscured by buttons). See Simulator for more information on testing with different display configurations, including testing light and dark mode.

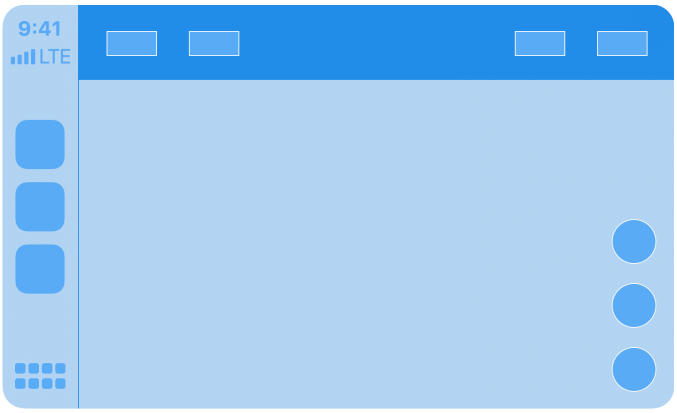
Base view

Map

The map template is a control layer that appears as an overlay over the base view and allows users to manipulate the map. It consists of a navigation bar and map buttons drawn as individual overlays. By default, the navigation bar appears when the user interacts with the app, and disappears after a period of inactivity. You can customize this behavior, including whether to hide the map buttons.

The navigation bar includes up to two leading buttons and two trailing buttons that can be specified with icons or text.

You can also specify up to four map buttons which are shown as icons. Use the map buttons to provide zooming and panning features. Although many cars support panning through direct manipulation of the car's touchscreen, there are cars that only support panning through knob or touch pad events. CarPlay supports these cars with a "panning mode." If your app supports any panning features, you must allocate one of the map buttons to be a pan button that allows the user to enter panning mode, and you must respond to the panning functions in `CPMapTemplate`.

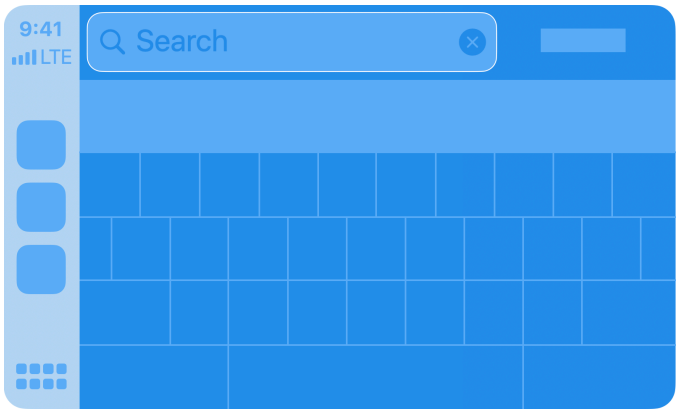


Map

Search

The search template displays a text entry field, a list of search results, and a keyboard. Your app parses the text by responding to `updatedSearchText` and updating the list of search results with an array of `CPListItem` elements. You must also take action when the user selects an item from the list by responding to `selectedResult`.

Note that many cars limit when the keyboard may be shown. See keyboard and list restrictions for details.



Search

Voice control

The voice control template allows you to provide visual feedback during a voice control session. CarPlay navigation apps can provide a voice control feature, but it must be restricted to navigation functions. In addition, navigation apps must display the voice control template whenever a voice control audio session is active.

The voice control template can only be used in navigation apps. Other CarPlay apps must use SiriKit or Siri Shortcuts to provide voice control features.

Voice control

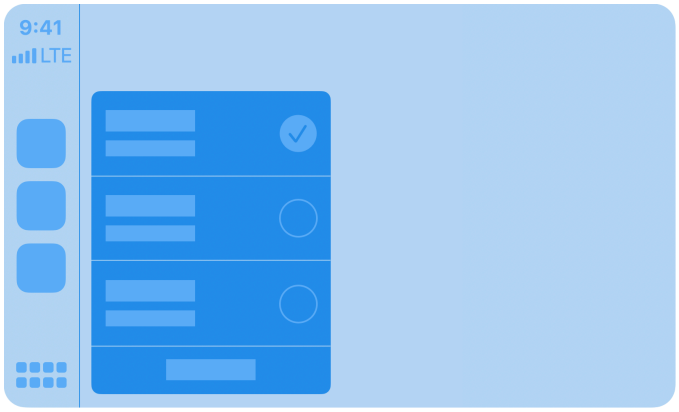
Panels

CarPlay navigation apps use panels to overlay information on the map. This includes trip previews, route selection, route guidance, and navigation alerts. You don't create panels directly. Instead, use the provided APIs to trigger them.

Trip preview panel. Display up to 12 potential destinations and select one. The trip preview panel is usually the result of a destination search. When users preview a trip, show a visual representation of that trip in your base view.

Trip preview

Route choice panel. Display potential routes for a trip and select one. Each route should have clear descriptions so the user can choose their preferred route. For example, a summary and optional description for a route could be "Via I-280 South" and "Traffic is light." When users preview a route, show a visual representation of that route in your base view.

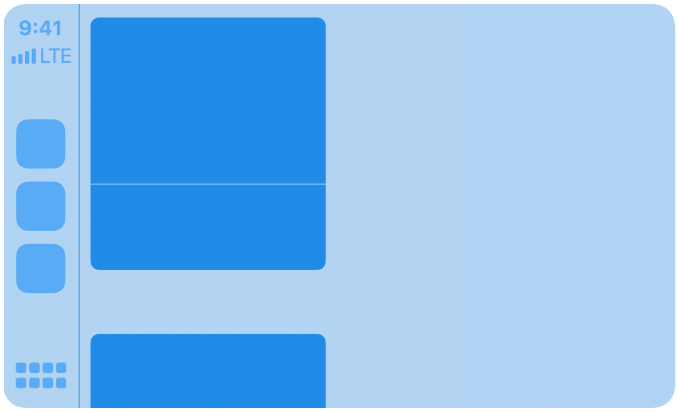


Route choice

Guidance and trip estimate panels. Display upcoming maneuvers and trip estimates.

Maneuvers are normally shown one at a time, but in cases where maneuvers appear in rapid succession, two maneuvers may be shown. The second maneuver may be repurposed to show lane guidance or a junction image for the first maneuver.

In addition to providing upcoming maneuvers, you should continuously update overall trip estimates.



Guidance and trip estimate

Each maneuver can include a symbol, instruction text, estimated remaining distance, and time.

You may optionally specify multiple variants for your images and instruction text so they appear differently in your app and the CarPlay Dashboard. This includes maneuver symbols, junction images, notification symbols, instruction text and notification text. To specify something different, use the dashboard variants of the properties—for example, by default `symbolImage` defines what appears in your app and the CarPlay Dashboard, but if you also specify a `dashboardSymbolImage` property, then it will be used in the CarPlay Dashboard.

Your app also provides metadata for maneuvers and lane guidance information that are displayed in the instrument cluster or HUD in supported vehicles. For details, see Show metadata in the instrument cluster or HUD.

Use the following guide when preparing maneuver symbol assets. Be sure to provide variants for light and dark interfaces.

	Maximum size in points	Maximum size in pixels (3x)	Maximum size in pixels (2x)
First maneuver symbol (symbol and instruction on one line)	50pt x 50pt	150px x 150px	100px x 100px
First maneuver symbol (symbol and instruction on two lines)	120pt x 50pt	360px x 150px	240px x 100px
Second maneuver symbol (symbol and instructions)	18pt x 18pt	54px x 54px	36px x 36px
Second symbol (symbol only)	120pt x 18pt	360px x 54px	240px x 36px
CarPlay Dashboard junction image	140pt x 100pt	420px x 300px	280px x 200px

Navigation alert panel. Display important, real time feedback and optionally give the user a chance to make a decision that will affect the current route. For example, you should show an alert if there is unexpected traffic ahead and you are recommending that the user take an alternate route. Navigation alerts result in a notification if your app is running in the background.

Navigation alerts can consist of an image, title, subtitle, duration for which the alert is visible before it's automatically dismissed, and up to 2 action buttons. For example, the action buttons could provide options to either maintain the current route, or take an alternate route. Starting in iOS 16 you can specify a navigation alert with longer subtitle text (in prior versions of iOS the subtitle is limited to 3 lines), no action buttons (in which case the alert will have a simple close button), or action buttons with custom colors.

Navigation alert

Startup

CarPlay navigation apps declare two CarPlay scenes, one for the main app window in CarPlay, and one for the CarPlay Dashboard. For details on how to set up a scene manifest that supports CarPlay, see Application scene manifest example.

Provide delegates for the CarPlay scene and the CarPlay Dashboard scene. Listen for the `didConnect` and `didDisconnect` methods to know when your app has been launched in each scene. In the main app window, your `CPTemplateApplicationSceneDelegate` will be called using the `didConnect` and `didDisconnect` methods that receive an interface controller and a window. `CPInterfaceController` and a `CPWindow` object.

For the main app view, retain references to both the interface controller and the map content window for the duration of the CarPlay session.

```
self.interfaceController = interfaceController
self.carWindow = window
```

Next, create a new view controller and assign it to the window's root view controller. Use the view controller to manage your map content as the base view in the window.

```
let rootViewController = MyRootViewController()
window.rootViewController = rootViewController
```

Finally, create a map template and assign it as the root template.

```
let rootTemplate: CPMapTemplate = createRootTemplate()
self.interfaceController?.setRootTemplate(rootTemplate, animated: false)
```

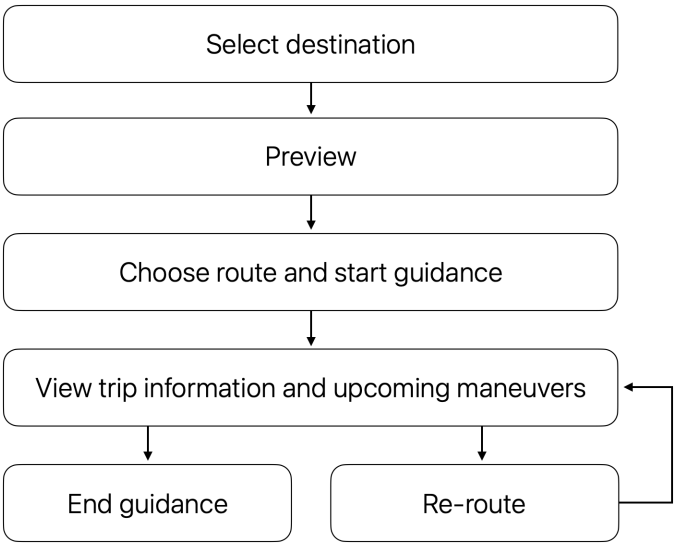
Create a default set of navigation bar buttons and map buttons and assign them to the root map template. Specify navigation bar buttons by setting up the `leadingNavigationBarButtons` and `trailingNavigationBarButtons` arrays. Specify map buttons by setting up the `mapButtons` array.

If your CarPlay navigation app supports panning, one of the buttons you create must be a pan button that lets the user enter panning mode. The pan button is essential in vehicles that don't support panning via the touch screen.

You can update the navigation bar buttons and map buttons dynamically based on the state of the app. For example, during active route guidance, you may choose to replace the default navigation bar buttons with an option to end route guidance.

Route guidance

All CarPlay navigation apps follow a standard flow for selecting a destination and providing route guidance.



Select destination. All route guidance starts with the user selecting a destination, whether that is the result of an on-screen search, voice command, or picking a category or destination from a list.

Preview. When a destination is selected, the user is shown a preview of the trip. At the same time, your map in the base view typically shows a visual representation of the trip. The preview also supports disambiguation when there are multiple matching destinations. For example, if the user chooses to navigate to a nearby park, the preview may show several parks to choose from.

Choose route and start guidance. Once the user has confirmed the destination, they may start route guidance. If there are multiple possible routes, your app can present the routes as options for the user to choose from.

View trip information and upcoming maneuvers. When the user starts route guidance, show real time information including upcoming maneuvers, and travel estimates (distance and time remaining) for the trip.

End guidance. Route guidance continues until the user arrives at the destination, or chooses to end route guidance.

Re-route. Your app can optionally return to an active guidance state with a new route.

Select destination

Use `CPInterfaceController` to present templates that allow the user to specify a destination. To present a new template, use `pushTemplate` with a supported `CPTemplate` class such as `CPGridTemplate`, `CPListTemplate`, `CPSearchTemplate`, or `CPVoiceControlTemplate`.

When the user selects an item or cancels the selection, your delegate will be called with information about the action that was taken.

You may present multiple templates in succession to support hierarchical selection. For example, you can show a list template that includes list items which lead to additional sublists when selected. Be sure to set `showsDisclosureIndicator` to `true` for list items that support hierarchical browsing, and push a new list template when the list item is selected. Hierarchical selections must never exceed five levels of depth.

Preview

After the user has selected a destination and you are ready to show trip previews, use `CPMapTemplate showTripPreviews` to provide an array of up to 12 `CPTrip` objects.

Each `CPTrip` object represents a journey consisting of an origin, a destination, up to 3 route choices, and estimates for remaining time and distance.

Use `CPRouteChoice` to define each route choice. Your descriptions for each route are provided as arrays of variable length strings in descending order of length (longest string first). CarPlay will display the longest string that fits in the available space on the screen.

For each `CPTrip`, be sure to provide travel estimates using `CPMapTemplate updateEstimates:` and update the estimates if the remaining time or distance change.

You may also customize the names of the start, overview, and additional routes buttons shown in the trip preview panel.

Choose route and start guidance

When the user selects a different route to preview, the delegate `selectedPreviewFor:` will be called. Respond by updating your map base view.

If the user decides to start a trip, the delegate `startedTrip:` will be called. Respond by starting route guidance. At this time, use `CPMapTemplate hideTripPreviews` to dismiss the trip preview panel.

```
mapTemplate.hideTripPreviews()
```

Next use `CPMapTemplate startNavigationSession` to start a navigation session for the selected trip and obtain a `CPNavigationSession` object that represents the active navigation session.

```
let session = mapTemplate.startNavigationSession(for: trip)
```

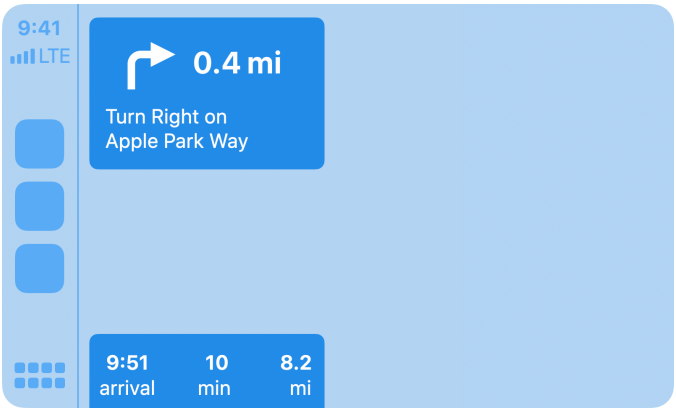
While you are calculating initial maneuvers, set the navigation session pause state to `CPTripPauseReasonLoading` so that CarPlay can display the correct state.

```
session.pauseTrip(for: .CPTripPauseReasonLoading)
```

At this time, update the navigation bar buttons and map buttons to provide appropriate actions for the user to manage their route.

View trip information and upcoming maneuvers

During turn by turn guidance, show route guidance information by updating `upcomingManeuvers` with information on upcoming turns. Each `CPManeuver` represents a single maneuver and may include a symbol, an instruction, metadata, and estimates for remaining time and distance.



Show a maneuver in the route guidance panel

Symbol. If the maneuver has an associated symbol, such as a turn right arrow, provide an image using `symbolSet`. The symbol will be shown in the route guidance card and any related notifications. You must provide two image variants using `CPImageSet`—one is used for rendering the symbol on light backgrounds, the other is used for rendering the symbol on dark backgrounds.

Instruction. Provide an instruction using `instructionVariants` which is an array of strings. Use the array to provide variants of different lengths so that CarPlay can display the instruction that best fits in the available space on the screen. For example, if the maneuver requires you to turn right on the street named "Solar Circle" you may choose to provide 3 instruction variants "Turn Right on Solar Circle," "Turn Right on Solar Cir.", and "Turn Right". CarPlay will display the instruction with the longest string length that fits in the available space. The array of instructions must be provided in descending order of length (longest string first). You may optionally provide `attributedInstructionVariants` to include embedded images in the instruction. This is useful if you need to display special symbols, such as a highway symbol, as part of the instruction. Note that other text attributes including text size and fonts will be ignored. If you provide `attributedInstructionVariants`, always provide text-only `instructionVariants` since CarPlay vehicles may not always support attributed strings.

Metadata. Provide maneuver type, maneuver state, junction type, traffic side, and lane guidance information for display in the instrument cluster or HUD of supported vehicles. For details, see Show metadata in the instrument cluster or HUD.

Add as many maneuvers as possible to `upcomingManeuvers`. At minimum, your app must maintain at least one upcoming turn in the `maneuvers` array at all times, and in cases where there are two maneuvers in quick succession, provide a second maneuver which may be shown on the screen simultaneously.

If you provide a second maneuver, you can customize its appearance by specifying a symbol style. In `CPMapTemplateDelegate`, return a `CPManeuverDisplayStyle` for the maneuver when requested. The display style only applies to the second maneuver.

If your app provides lane guidance information, you must use the second maneuver to show lane guidance. Create a second maneuver containing `symbolSet` with dark and light images that occupy the full width of the guidance panel (maximum size 120pt x 18pt), provide an empty array for `instructionVariants`, and in the `CPMapTemplateDelegate`, return a symbol style of `CPManeuverDisplayStyleSymbolOnly` for the maneuver.

Show a maneuver with lane guidance information

Your app is responsible for continuously updating estimates for remaining time and distance for each maneuver, and for the overall trip. Use `CPNavigationSession updateEstimates:` to update estimates for each maneuver, and `CPMapTemplate updateEstimates` to update overall estimates for the trip. Only update the values when significant changes occur, such as when the number of remaining minutes changes.

If you need to display an alert related to the map or navigation, create a `CPNavigationAlert` and use `CPMapTemplate present` to show it. Navigation alerts can be configured to automatically disappear after a fixed interval. They may also be shown as a notification, even when your app is not in the foreground.

For each maneuver and navigation alert, specify whether it should be shown as a CarPlay notification when your app is running in the background. Respond to the `shouldShowNotificationFor` delegate call to specify the maneuver or navigation alert behavior. In the case of a maneuver, you can optionally include updating travel estimates as part of the notification.

In addition to the route guidance panel, maneuvers may also be shown in notifications, or sent to vehicles that support the display of CarPlay metadata in their instrument cluster or heads up display.

End guidance

When route guidance is paused, canceled, or finished, call the appropriate method in `CPNavigationSession`.

In some cases, CarPlay route guidance may be canceled by the system. For example, if the car's native navigation system starts route guidance, CarPlay route guidance automatically terminates. In this case, your delegate will receive `mapTemplateDidCancelNavigation` and you should end route guidance immediately.

Re-route

Starting in iOS 17.4, your app can programmatically return to an active guidance state. Use the `CPNavigationSession` method `resumeTrip` and provide a `CPRouteInformation` object with details about the new route.

Keyboard and list restrictions

Some cars limit keyboard use and the lengths of lists while driving. iOS automatically disables the keyboard and reduces list lengths when the car indicates it should do so. However, if your app needs to adjust other user interface elements in response to these changes, you can receive notifications when the limits change. For example, you may want to disable a keyboard icon or adjust list items when list lengths are shorter. Use `CPSessionConfiguration` to observe `limitedUserInterfaces`.

Voice prompts

Voice prompts are essential for a route guidance experience, but you must ensure that your app is a good audio citizen and works well with other audio sources on iPhone and in the car.

Audio session configuration

CarPlay navigation apps must use the following audio session configuration when playing voice prompts for upcoming maneuvers.

1. Set the audio session category to `AVAudioSessionCategoryPlayback`.
2. Set the audio session mode to `AVAudioSessionModeVoicePrompt`.
3. Set the audio session category options to `AVAudioSessionCategoryOptionInterruptSpokenAudioAndMixWithOthers` and `AVAudioSessionCategoryOptionDuckOthers`.

Voice prompts are played over a separate audio channel and mixed with audio sources in the car, including the car's own audio sources such as FM radio.

`AVAudioSessionCategoryOptionInterruptSpokenAudioAndMixWithOthers` allows voice prompts to pause certain apps with spoken audio (such as podcasts or audio books) and mix with other apps such as music.

`AVAudioSessionCategoryOptionDuckOthers` allows voice prompts to duck (lower the volume) for other apps such as music while your audio is played.

Activate and deactivate the audio session

Keep your audio session deactivated until you are ready to play a voice prompt. Call `setActive` with `YES` only when a voice prompt is ready to play. You may keep the audio session active for short durations if you know that multiple audio prompts are going to be played in rapid succession. However, while your `AVAudioSession` is active, music apps will remain ducked, and apps with spoken audio will remain paused. Don't hold on to the active state for more than few seconds if audio prompts are not playing.

When you are done playing a voice prompt, call `setActive` with `NO` to allow other audio to resume.

Prompt style

In some cases it doesn't make sense to play a voice prompt. For example, the user may be on a phone call or in the middle of using Siri.

Just before playing each voice prompt, check the audio session's `promptStyle`. If necessary, it will return a hint to alter the type of prompt you should play in response to other system audio.

Prompt style	Action
`None`	Don't play any sound
`Short`	Play a tone
`Normal`	Play a full spoken prompt

Show second map in CarPlay Dashboard or the instrument cluster

People using your navigation app want to see important information, even when your app is not the foreground app in CarPlay.

Support for CarPlay Dashboard. Starting with iOS 13.4, you can add support for CarPlay Dashboard. Display your map, upcoming maneuvers, and dashboard buttons so they are available at a glance inside CarPlay Dashboard.

Support for the instrument cluster. Starting with iOS 15.4, you can add support for instrument cluster displays in supported vehicles. Display your map and upcoming maneuvers, so they are visible at a glance in the car's instrument cluster display.

It's easy to support both CarPlay Dashboard and instrument cluster displays since they work in the same way.

`CPTemplateApplicationDashboardScene` and `CPTemplateApplicationInstrumentClusterScene` are new `UIScene` subclasses that CarPlay creates when it determines that your app should appear in CarPlay Dashboard or the instrument cluster.

`CPDashboardController` and `CPDashboardButton` let you manage the CarPlay Dashboard and the buttons that appear inside it. `CPInstrumentClusterController` lets you manage instrument cluster displays.

Indicate support for the CarPlay dashboard and instrument cluster

In your application scene manifest, set `CPSupportsDashboardNavigationScene` and `CPSupportsInstrumentClusterNavigationScene` to true and provide corresponding keys for your scenes and delegates. Also see Application scene manifest example.

Create scene delegates

Define delegates for CarPlay Dashboard and instrument cluster scenes just like you would for the main template application scene. These delegates conform to `CPTemplateApplicationDashboardSceneDelegate` and `CPTemplateApplicationInstrumentClusterSceneDelegate` and will be called with instances of `CPDashboardController` or `CPInstrumentClusterController`.

Draw your content

Use the provided windows to draw map content for display in the CarPlay Dashboard or instrument cluster.

When drawing maps in the instrument cluster, you must follow these guidelines:

- Draw a minimal version of your map with minimal clutter
- Show a detailed view of the upcoming route, not an overview
- Ensure the current heading is facing up (the top of the screen)

Also, as with all maps rendered in CarPlay, be sure to observe safe areas, and light and dark mode settings (similar to your base view, use the `contentMode` in `CPTemplateApplicationDashboardScene` or `CPTemplateApplicationInstrumentClusterScene`).

When navigation begins in your app using `CPMapTemplate` and `CPNavigationSession`, CarPlay automatically displays maneuver information.

For the CarPlay Dashboard, you can also provide two instances of `CPDashboardButton` to `CPDashboardController`. These buttons appear in the guidance card area when your app is not actively navigating. People can interact with your app through the dashboard buttons as well as within your main app interface.

For instrument cluster displays, some cars may allow users to zoom the map in and out. It's your responsibility to respond to these events in your delegate. Similarly, if your app includes a compass or speed limit, the corresponding delegates will tell your app whether it's appropriate to draw them or not. Depending on the shape of the car's instrument cluster, your view area may be partially obscured by other elements in the car. Override `viewSafeAreaInsetsDidChange` on your view controller to know when the safe area changes, and use the `safeAreaLayoutGuide` on your cluster view to ensure that important content in the area of the view is always visible.

Show metadata in the instrument cluster or HUD

People using your navigation app want to see important information in the instrument cluster or HUD (Head-Up Display) in supported vehicles. Many vehicles, even those without a full digital instrument cluster display, show metadata for upcoming maneuvers in smaller displays inside their instrument cluster. Modern vehicles with a HUD also show metadata on their windshield.

Starting with iOS 17.4, your app can provide metadata for upcoming maneuvers. This includes maneuver state, maneuver type (e.g. “turn right”, “make a U-turn”), junction type, and lane guidance information.

Declare support for metadata. Use the delegate method `mapTemplateShouldProvideNavigationMetadata` in `CPMapTemplateDelegate` to indicate that your app supports sending metadata to the vehicle.

Provide information about upcoming maneuvers and the current trip. Supply multiple maneuvers, including maneuver type and lane guidance information, when route guidance starts. Use `add CPManeuver` and `add CPLaneGuidance`.

Provide as many maneuvers as possible to support vehicles that display multiple maneuvers in the instrument cluster or HUD, and to improve performance. Additional maneuvers can be added during route guidance.

Your app should also set the current road name, and update the maneuver state which indicates progress within a maneuver. When approaching a maneuver, the maneuver state should transition from `continue` → `initial` → `prepare` → `execute` → `continue`.

Maneuver state	Description
`continue`	Continue along this route until next maneuver
`initial`	Maneuver is in the near future
`prepare`	Maneuver is in the immediate future
`execute`	In maneuver

Required maneuver properties include `maneuverType`, `junctionType` and `trafficSide`. Note that maneuver metadata supplements the `symbol` and `instruction` which is used on the CarPlay screen.

For lane guidance, use `CPLaneGuidance` and `CPLane` to provide lane guidance metadata for the vehicle. Again, lane guidance metadata supplements showing lane guidance using `symbolSet` on the CarPlay screen.

Manuever types

For the maneuver type, select from one of the following predefined values.

Maneuver type	Description
`arriveAtDestination`	Destination has been reached, navigation will end
`arriveAtDestinationLeft`	Destination has been reached; it is on the left and navigation will end
`arriveAtDestinationRight`	Destination has been reached; it is on the right and navigation will end
`arriveEndOfDirections`	Navigation has completed, but the rest of the journey will need to use another transport method
`arriveEndOfNavigation`	Navigation has completed, but the rest of the journey will need to use another transport method
`changeFerry`	Change to a different ferry
`changeHighway`	Highway to highway change with implied or unknown side of the road
`changeHighwayLeft`	Highway to highway change from the left side of the road
`changeHighwayRight`	Highway to highway change from the right side of the road
`enterRoundabout`	Enter roundabout
`enter_Ferry`	Enter ferry
`exitFerry`	Exit ferry
`exitRoundabout`	Exit roundabout
`followRoad`	Continue to follow the road the vehicle is currently on
`highwayOffRampLeft`	Exit highway on left
`highwayOffRampRight`	Exit highway on right

`keepLeft`	Usually for bifurcations or other smooth maneuvers (compare to `slightLeftTurn`)
`keepRight`	Usually for bifurcations or other smooth maneuvers (compare to `slightRightTurn`)
`leftTurn`	Angle is between -45° and -135°
`leftTurnAtEnd`	At the end of the road, turn left
`noTurn`	No turn (default value)
`offRamp`	Take ramp to leave highway
`onRamp`	Take ramp to merge onto highway
`rightTurn`	Angle is between 45° and 135°
`rightTurnAtEnd`	At the end of the road, turn right
`roundaboutExit1`	Exit roundabout at the 1st street
`roundaboutExit10`	Exit roundabout at the 10th street
`roundaboutExit11`	Exit roundabout at the 11th street
`roundaboutExit12`	Exit roundabout at the 12th street
`roundaboutExit13`	Exit roundabout at the 13th street
`roundaboutExit14`	Exit roundabout at the 14th street
`roundaboutExit15`	Exit roundabout at the 15th street
`roundaboutExit16`	Exit roundabout at the 16th street
`roundaboutExit17`	Exit roundabout at the 17th street
`roundaboutExit18`	Exit roundabout at the 18th street
`roundaboutExit19`	Exit roundabout at the 19th street
`roundaboutExit2`	Exit roundabout at the 2nd street
`roundaboutExit3`	Exit roundabout at the 3rd street
`roundaboutExit4`	Exit roundabout at the 4th street

`roundaboutExit5`	Exit roundabout at the 5th street
`roundaboutExit6`	Exit roundabout at the 6th street
`roundaboutExit7`	Exit roundabout at the 7th street
`roundaboutExit8`	Exit roundabout at the 8th street
`roundaboutExit9`	Exit roundabout at the 9th street
`sharpLeftTurn`	Angle is between -135° and -180°
`sharpRightTurn`	Angle is between 135° and 180°
`slightLeftTurn`	Turn onto a different road (compare to `keepLeft`)
`slightRightTurn`	Turn onto a different road (compare to `keepRight`)
`startRoute`	Proceed to the beginning of the route
`startRouteWithUTurn`	Make a U-turn and proceed to the route
`straightAhead`	Continue straight through the intersection (implies a road name will change)
`uTurn`	Make a U-turn and proceed to the route
`uTurnAtRoundabout`	Use roundabout to make a U-turn
`uTurnWhenPossible`	Make a U-turn when possible

Junction types

For the junction type, select from one of the following predefined values.

`intersection`	Single intersection with junction elements representing roads coming to a common point
`roundabout`	Roundabout with junction elements representing roads exiting the roundabout

Traffic side

For the traffic side, select from one of the following predefined values.

`right`	Right (or anti-clockwise for roundabouts)
`left`	Left (or clockwise for roundabouts)

Lane angles and lane status

Lane angles specify angles (or a single angle) between -180° and +180°. For the lane status, select from one of the following predefined values.

`notGood`	The vehicle should not take this lane
`good`	The vehicle can take this lane, but may need to move lanes again before upcoming maneuvers
`preferred`	The vehicle should take this lane to be in the best position for upcoming maneuvers

Test your navigation app

If you are developing a navigation app, it's important to try different display configurations to ensure your map drawing code works correctly. Note that CarPlay supports both landscape and portrait displays and can scale from 2x at low resolutions to 3x at high resolutions. Here are some recommended screen sizes to test.

	Width and height	Scale
Minimum (smallest possible CarPlay screen)	748px x 456px	2.0
Standard (default resolution typical of many CarPlay screens)	800px x 480px	2.0
High resolution (typical of larger CarPlay screens)	1920px x 720px	3.0
Portrait (example of a vertical CarPlay screen)	900px x 1200px	3.0

In CarPlay Simulator, simply click **Configure** to change the display configuration.

In Xcode Simulator, first enable extra options by entering the following command in Terminal before launching Xcode Simulator. Xcode Simulator does not simulate the instrument cluster or show metadata.

```
defaults write com.apple.iphonesimulator CarPlayExtraOptions -bool YES
```

Test maps in instrument cluster displays

Use CarPlay Simulator to test your map in instrument cluster displays.

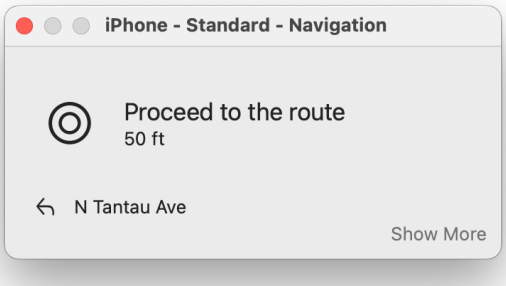
Click **Configure** | **Cluster Display**, turn on **Instrument Cluster Display enabled** and specify scale factor, screen size, safe area, and safe area sizes. Here are some recommended configurations to test.

	Scale Factor	Size	Safe Area Origin	Safe Area Size
Minimum	3x	300 x 200	0, 0	300 x 200
Basic	2x	640 x 480	0, 0	640 x 480
Widescreen (wide safe area)	3x	1920 x 720	420, 0	1080 x 720
Widescreen (small safe area)	2x	1920 x 720	640, 120	640 x 480

Test metadata in the instrument cluster or HUD

Use CarPlay Simulator to confirm that your app is correctly supplying metadata for display in the instrument cluster or HUD in supported vehicles.

With an active navigation session, click **Navigation** to view the next upcoming maneuver.



Click **Show More** to see the full sequence of upcoming maneuvers provided by your app. Inside the larger information screen, click the table icons to Maneuvers or Lane Guidances to view detailed information.

Application scene manifest example

The following is an example of an application scene manifest that supports both the CarPlay Dashboard and instrument cluster displays.

```
<key>UIApplicationSceneManifest</key>
<dict>
    <!-- Indicate support for CarPlay dashboard -->
    <key>CPSupportsDashboardNavigationScene</key>
    <true/>
    <!-- Indicate support for instrument cluster displays -->
    <key>CPSupportsInstrumentClusterNavigationScene</key>
    <true/>
    <!-- Indicate support for multiple scenes -->
    <key>UIApplicationSupportsMultipleScenes</key>
    <true/>
    <key>UISceneConfigurations</key>
    <dict>
        <!-- For device scenes -->
        <key>UIWindowSceneSessionRoleApplication</key>
        <array>
            <dict>
                <key>UISceneClassName</key>
                <string>UIWindowScene</string>
                <key>UISceneConfigurationName</key>
                <string>Phone</string>
                <key>UISceneDelegateClassName</key>
                <string>MyAppWindowSceneDelegate</string>
            </dict>
        </array>
       <!-- For the main CarPlay scene -->
        <key>CPTemplateApplicationSceneSessionRoleApplication</key>
        <array>
            <dict>
                <key>UISceneClassName</key>
                <string>CPTemplateApplicationScene</string>
                <key>UISceneConfigurationName</key>
                <string>CarPlay</string>
                <key>UISceneDelegateClassName</key>
```

```
            <string>MyAppCarPlaySceneDelegate</string>
        </dict>
    </array>
    <!-- For the CarPlay Dashboard scene -->
    <key>CPTemplateApplicationDashboardSceneSessionRoleApplication</key>
    <array>
        <dict>
            <key>UISceneClassName</key>
            <string>CPTemplateApplicationDashboardScene</string>
            <key>UISceneConfigurationName</key>
            <string>CarPlay-Dashboard</string>
            <key>UISceneDelegateClassName</key>
            <string>MyAppCarPlayDashboardSceneDelegate</string>
        </dict>
    </array>
    <!-- For the CarPlay instrument cluster scene -->
    <key>CPTemplateApplicationInstrumentClusterSceneSessionRoleApplication</key>
    <array>
        <dict>
            <key>UISceneClassName</key>
            <string>CPTemplateApplicationInstrumentClusterScene</string>
            <key>UISceneConfigurationName</key>
            <string>CarPlay-Instrument-Cluster</string>
            <key>UISceneDelegateClassName</key>
            <string>MyAppCarPlayInstrumentClusterSceneDelegate</string>
        </dict>
    </array>
</dict>
</dict>
```

Sample code

The following sample code is available to help you get started developing your CarPlay app.

Integrating CarPlay with your music app

CarPlay Music is a sample music app that demonstrates how to display a custom UI from a CarPlay–enabled vehicle. CarPlay Music integrates with the CarPlay framework by implementing the `CPNowPlayingTemplate` and `CPListTemplate`. This sample's iOS app component provides a logging interface to help you understand the life cycle of a CarPlay app, as well as a music controller.

Download

Integrating CarPlay with your quick food ordering app

CarPlay Quick-Ordering is a sample quick food ordering app that demonstrates how to display custom ordering options in a vehicle that has CarPlay enabled. The sample app integrates with the CarPlay framework by implementing `CPTemplate` subclasses, such as `CPPointOfInterestTemplate` and `CPListTemplate`. This sample's iOS app component provides a logging interface to help you understand the life cycle of a CarPlay app.

Download

Integrating CarPlay with your navigation app

Coastal Roads is a sample navigation app that demonstrates how to display a custom map and navigation instructions in a CarPlay–enabled vehicle. Coastal Roads integrates with the CarPlay framework by implementing the map and additional `CPTemplate` subclasses, such as `CPGridTemplate` and `CPListTemplate`. This sample's iOS app component provides a logging interface to help you understand the life cycle of a CarPlay app.

Download

Publish your CarPlay app

When you are ready to publish your CarPlay app on the App Store, follow the same process as for any iOS app and use App Store Connect to submit your app.

Ensure that your app follows the CarPlay App Guidelines.

Apple Inc.
One Apple Park Way
Cupertino, CA 95014
408-996-1010