

CarPlay App Programming Guide

March 31, 2021



Table of Contents

Introduction	3
Overview	4
CarPlay app entitlement	4
CarPlay app guidelines	5
Development environment	8
Configure your entitlement	8
CarPlay audio apps and backward compatibility	10
CarPlay communication apps and backward compatibility.....	10
Simulator.....	11
Templates	14
Action sheet	15
Alert.....	15
Contact.....	16
Grid.....	16
Information	17
List.....	18
Now playing.....	19
Point of interest.....	20
Tab bar.....	21
Assets	22
Build your CarPlay app.....	24
Startup	24
Create a list template	26
Create a now playing template.....	27
Play audio	28
Work while iPhone is locked	28
CarPlay navigation apps	29
Templates	29
Base View	29
Map.....	30
Search	31

Voice control	32
Panels	33
Trip preview	33
Route choice	34
Guidance and trip estimate.....	35
Navigation alert.....	37
Startup	38
Route guidance.....	39
Select destination	39
Preview	40
Choose route and start guidance	40
Show trip information and upcoming maneuvers.....	41
End guidance	43
Keyboard and list restrictions	43
Voice prompts.....	43
Audio session configuration	43
Activate and deactivate the audio session	44
Prompt style.....	44
CarPlay Dashboard.....	45
Create a scene delegate	45
Draw your CarPlay Dashboard content.....	45
Dashboard scene manifest	46
Publish your CarPlay app	48

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 categories of apps:

- Audio
- Communication
- EV charging
- Navigation
- Parking
- Quick food ordering

This guide describes how to create a CarPlay app.

Overview

Users download CarPlay apps from the App Store and use them on iPhone like any other app. When an iPhone with a CarPlay app is 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 an existing app.

CarPlay apps are designed to look and feel like your app on iPhone, but with UI elements that are similar to the 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 Apps](#).

CarPlay app entitlement

All CarPlay apps require a CarPlay app entitlement.

To request a CarPlay app entitlement, go to <http://developer.apple.com/carplay> and provide information about your app, including the CarPlay app category. You must also 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 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 audio apps

1. Never show song lyrics on the CarPlay screen.
2. Text to speech systems (web readers, email readers, etc.) are not permitted.

Additional guidelines for communication apps

1. Communication apps must provide short form text messaging (not email) features, VoIP calling features, or both.
2. Messaging must be handled using SiriKit. Your app must support sending a message, searching for messages, and setting attributes on a message.
3. VoIP calling must be handled using CallKit and SiriKit. Your app must support starting an audio call, and searching the user's call history.

Additional guidelines for 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 navigation apps

1. 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.
2. 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.
3. 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.
4. 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.
5. Correctly handle audio. Voice prompts must work concurrently with the vehicle's audio system (such as FM radio) and your app should not needlessly activate audio sessions when there is no audio to play.
6. Ensure that your map is appropriate in each supported country.
7. Be open and responsive to feedback. Apple may contact you in the event that Apple or automakers have input to design or functionality.
8. Voice control must be limited to navigation features.

Additional guidelines for parking apps

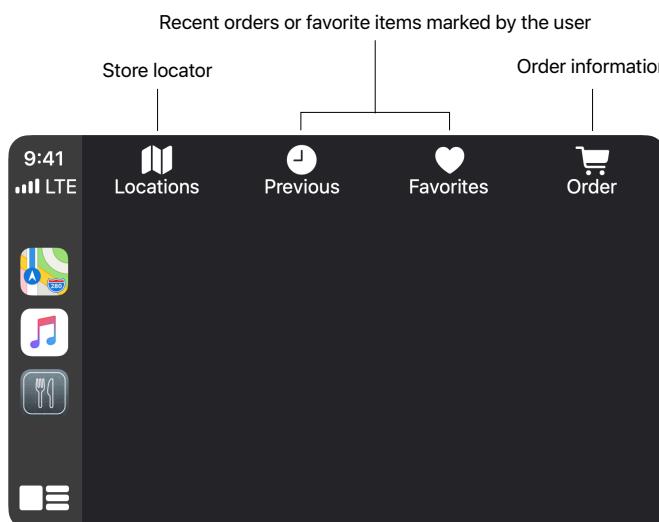
1. Parking apps must provide meaningful functionality relevant to driving (e.g. your app can't just be a list of parking locations).
2. When showing locations on a map, do not expose locations other than parking.

Additional guidelines for 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. The category is not intended for 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.

Configure your entitlement

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. Select **Certificates, Identifiers & Profiles** and click **Identifiers** on the sidebar.
3. Select the App ID associated with your app, or create a new App ID.
4. Scroll down to **Additional Capabilities** near the bottom of the page.
5. Enable all necessary CarPlay App capabilities 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 **Reference | Provisioning with Managed Capabilities | App Identifier** in Developer Account Help.

<https://help.apple.com/developer-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 that only supports the CarPlay framework.

```
<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.

Use the entitlement keys that match your selected provisioning profile.

Entitlement	Key
CarPlay Audio App (CarPlay framework) App supports the CarPlay framework. Include both CarPlay audio app entitlements if your app supports the CarPlay framework and the Media Player framework. Requires iOS 14 or later.	com.apple.developer.carplay-audio
CarPlay Audio App (Media Player framework) App supports the Media Player framework (deprecated). Include both CarPlay audio app entitlements if your app supports the CarPlay framework and the Media Player framework.	com.apple.developer.playable-content
CarPlay Communication App App supports the CarPlay framework, CallKit, SiriKit messaging intents, and SiriKit VoIP calling intents. Requires iOS 14 or later.	com.apple.developer.carplay-communication
CarPlay Messaging App App supports SiriKit messaging intents.	com.apple.developer.carplay-messaging
CarPlay VoIP Calling App App supports CallKit and SiriKit VoIP calling intents.	com.apple.developer.carplay-calling
CarPlay EV Charging App App supports the CarPlay framework. Requires iOS 14 or later.	com.apple.developer.carplay-charging
CarPlay Navigation App App supports the CarPlay framework. Requires iOS 12 or later.	com.apple.developer.carplay-maps
CarPlay Parking App App supports the CarPlay framework. Requires iOS 14 or later.	com.apple.developer.carplay-parking
CarPlay Quick Food Ordering App App supports the CarPlay framework. Requires iOS 14 or later.	com.apple.developer.carplay-quick-ordering

[CarPlay audio apps and backward compatibility](#)

Starting with iOS 14, CarPlay audio apps may use the CarPlay framework to present a customized user interface. CarPlay audio apps that use the CarPlay framework must include the com.apple.developer.carplay-audio entitlement.

CarPlay audio apps that use the Media Player framework must include the com.apple.developer.playable-content entitlement. If you want your CarPlay audio app to support iOS 13 and earlier, you must support the Media Player framework and include this entitlement. CarPlay apps that support the Media Player framework will continue to work on iOS 14.

CarPlay audio apps can support the CarPlay framework, Media Player framework, or both. Be sure to include the correct entitlement (or include both entitlements) to match the framework(s) you are using.

[CarPlay communication apps and backward compatibility](#)

Starting with iOS 14, CarPlay communication apps may use the CarPlay framework to present a user interface. CarPlay communication apps that use the CarPlay framework must include the com.apple.developer.carplay-communication entitlement.

The CarPlay communication app category replaces the older CarPlay messaging app and CarPlay VoIP calling app categories. You should include the com.apple.developer.carplay-messaging and com.apple.developer.carplay-calling entitlements to support iOS 13 and earlier. On iOS 13 and earlier your app will not be able to present a UI, but the user can still interact with your app using SiriKit and CallKit.

Only include the com.apple.developer.carplay-communication entitlement if you use the CarPlay framework.

Simulator

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

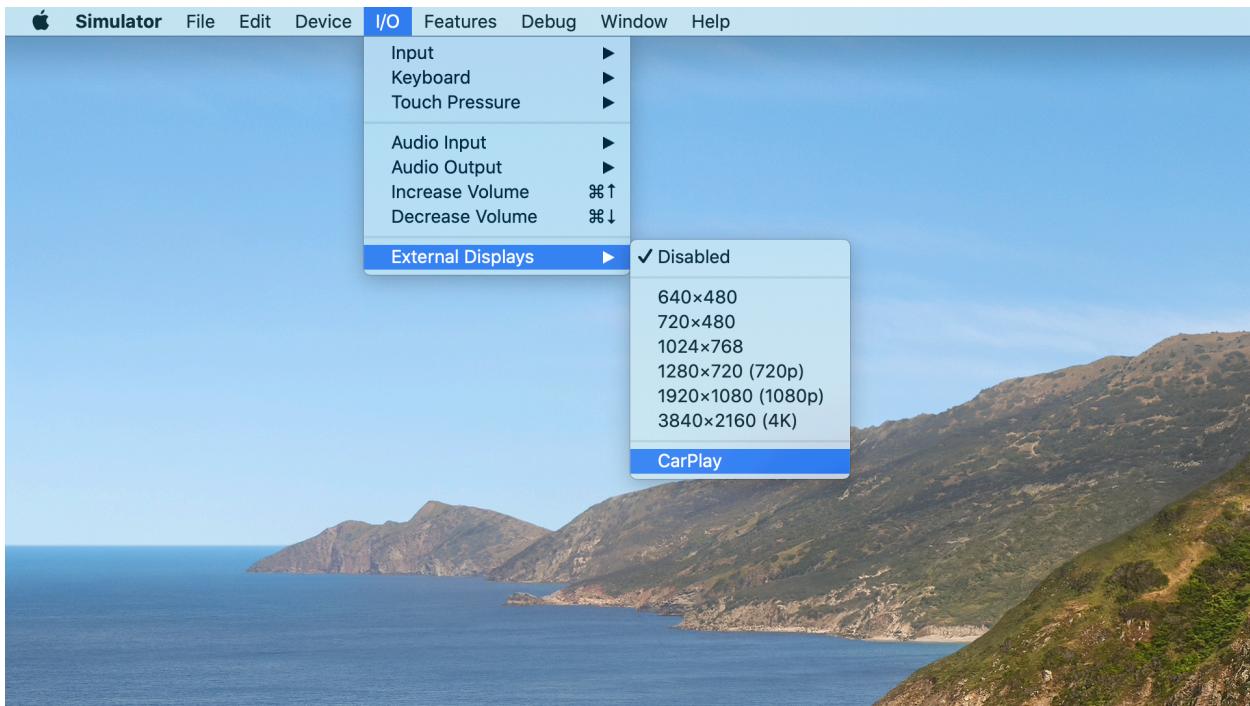
While Simulator is useful during development, certain CarPlay features are not available in Simulator and you should not rely on it as the sole method to develop your app.

Some features that cannot be tested in Simulator:

- **Testing while iPhone is locked.** Most users interact with CarPlay while iPhone is locked. Your app should perform normally even when iPhone is locked.
- **Testing Siri.** CarPlay relies heavily on Siri. Ensure that your app works as expected during Siri interactions.
- **Testing audio behavior.** Your app must be a good audio citizen. Remember that other audio sources may be playing while CarPlay is active. Deactivate your audio session except when you are actually playing audio. For example, when the car's FM radio is playing, navigation voice prompts should cause the FM radio volume level to be temporarily lowered, then raised after the voice prompt has finished playing.

It is highly recommended to develop and test CarPlay apps using a car or aftermarket system that supports wireless CarPlay. This will allow you to run CarPlay wirelessly while iPhone is simultaneously connected to Xcode on your Mac using a Lightning to USB cable.

To access CarPlay in Simulator, select I/O, External Displays, and CarPlay to show the CarPlay screen.



Show CarPlay in a second window

If you are developing a CarPlay navigation app, you can configure Simulator with extra options. To enable extra options, enter the following command in Terminal before launching Simulator.

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

If you configure Simulator with extra options, you can modify the CarPlay screen width, height, and scale.

By default, the CarPlay screen is configured as a display that is 800 pixels wide and 480 pixels high. In this configuration CarPlay will use a scale of 2.0. This is typical of many CarPlay systems. At higher resolutions, CarPlay automatically uses a scale of 3.0.

It is recommended that you test your CarPlay navigation app using the following settings. In particular, you need to ensure that your base map renders correctly at all resolutions and scales.

	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)	768px x 1024px	2.0

Templates

CarPlay apps are built from a fixed set of user interface 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).

CarPlay supports general purpose templates such as alerts, lists, and tab bars. It also supports templates designed for specific features such as contacts, maps, and a now playing screen.

Each CarPlay app category may access a limited set of templates. Access to templates is determined by your app entitlement. For example, an audio app may only use alert, grid, list, now playing, and tab bar templates.

If you attempt to use a template not supported by your entitlement, an exception will occur at runtime.

	Audio	Communication	EV charging	Navigation	Parking	Quick food ordering
Action Sheet		●	●	●	●	●
Alert	●	●	●	●	●	●
Contact		●		●		
Grid	●	●	●	●	●	●
Information		●	●	●	●	●
List	●	●	●	●	●	●
Map				●		
Now Playing	●					
Point of Interest			●		●	●
Search				●		
Tab bar	●	●	●	●	●	●
Voice control				●		

There is a limit to the number of templates that you can push onto the screen. All apps are limited to pushing up to 5 templates in depth, including the root template. Quick food ordering apps are limited to 2 templates in depth.

Action sheet

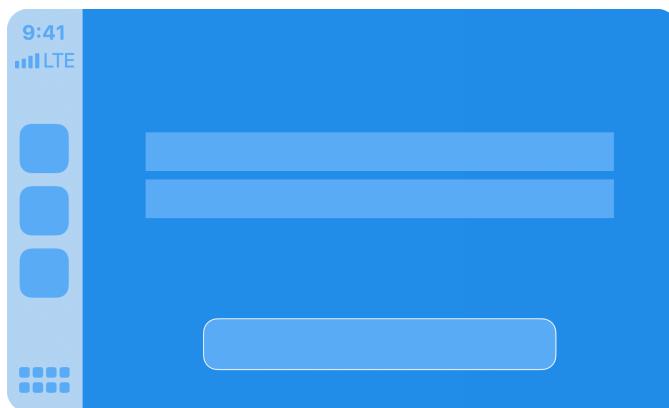
An action sheet is a specific style of alert that appears in response to a control or action, and presents a set of two or more choices related to the current context. Use action sheets to let people initiate tasks, or to request confirmation before performing a potentially destructive operation. The action sheet template can be used in communication, EV charging, navigation, parking, and quick food ordering apps.



Action sheet

Alert

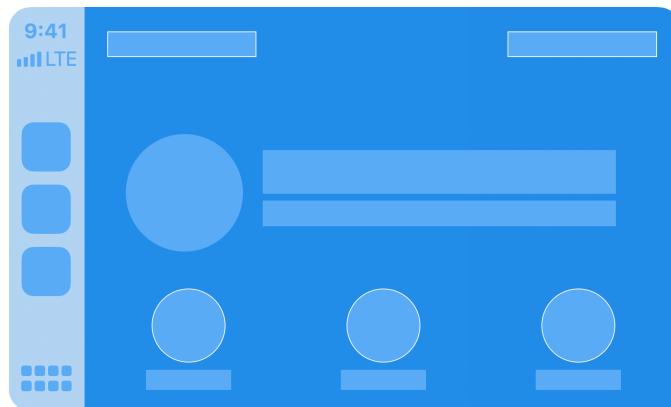
Alerts convey important information related to the state of your app. An alert consists of a title and one or more buttons. You can provide titles of varying lengths and let CarPlay choose the title that best fits the available screen space. If underlying conditions permit, alerts can be dismissed programmatically.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



Alert

Contact

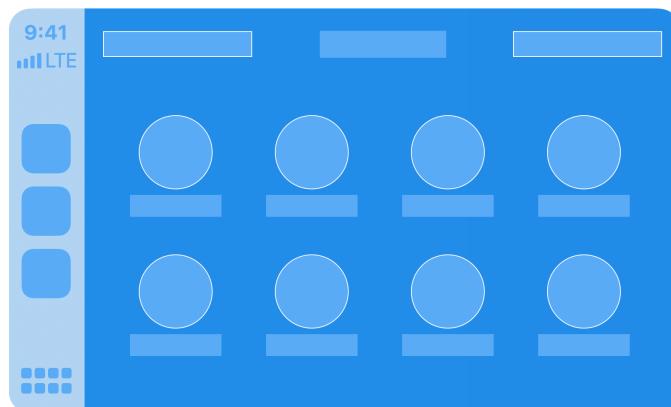
Contacts allow you to present information about a person or business. A contact consists of an image, title, subtitle, and action buttons. Use action buttons to let users perform tasks related to the current contact, such as making a phone call or sending a message. The contact template can be used in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



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 categories. The grid also includes a navigation bar with a title, leading buttons, and trailing buttons which can be shown as icons or text. The grid template can be used by all apps.



Grid

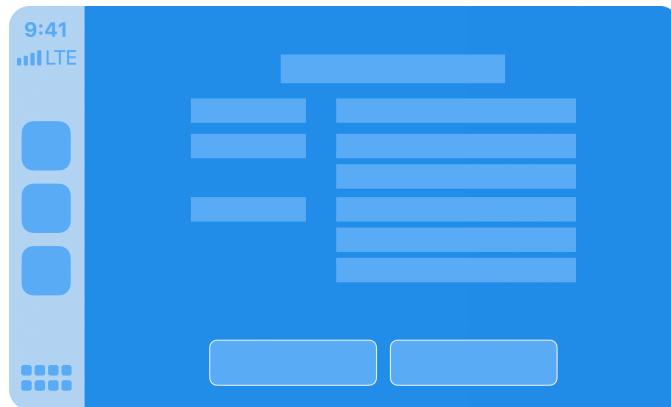
Information

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.

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.

The information template can be used by communication, EV charging, parking, navigation, and quick food ordering apps.



Information

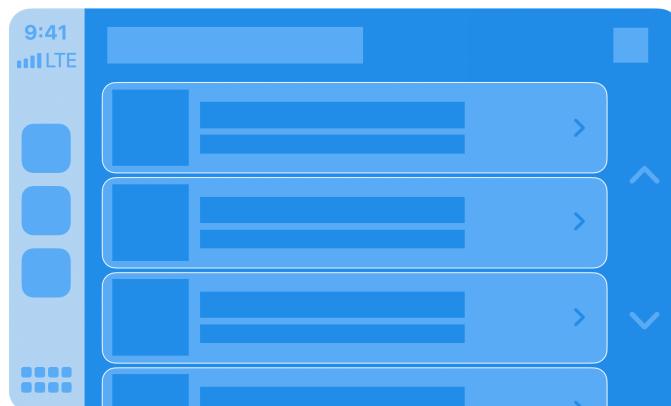
List

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 for audio apps, or the messages list item for communication apps.

Some cars dynamically limit lists to a maximum of 12 items. Your app can always check the currently permitted maximum, but you always need to be prepared to handle the case where only 12 items can be shown. Items beyond the maximum will not be shown.

Lists can be used by all apps.



List



List with an image row list item for audio apps

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.

The now playing template can be used in audio apps.



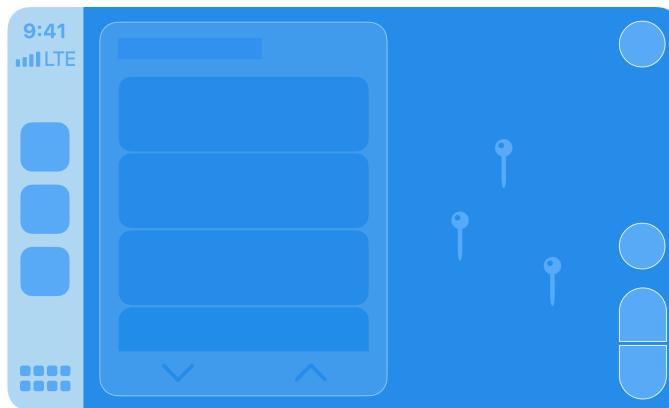
Now playing

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. The list of locations should be limited to those that are most relevant or nearby.

The point of interest template can be used by EV charging, parking, and quick food ordering apps.



Point of interest

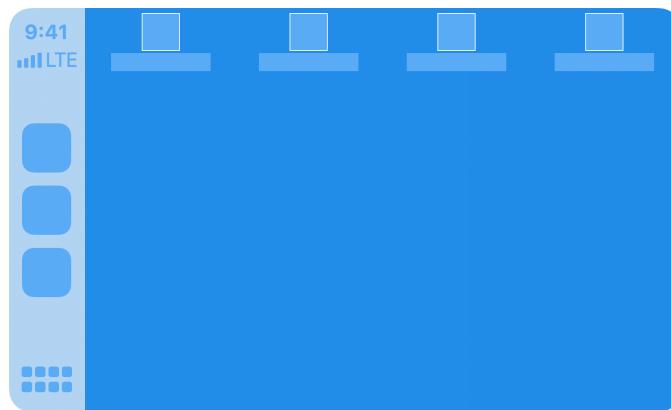
Tab bar

The tab bar is a versatile container for other templates, where each template occupies one tab in the tab bar.

Users can use the tab bar to rapidly switch between different templates.

The tab bar can have up to 5 tabs, although this may change in the future so you should not rely on this number. Instead, observe the maximum tab count provided by iOS. In the case of audio apps, one tab is always occupied by the now playing button.

The tab bar template can be used by all apps.



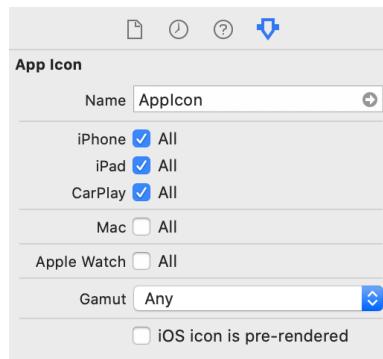
Tab bar

Assets

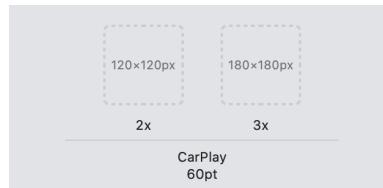
CarPlay assets include your app icon and images used in templates such as icons and buttons. Create assets for CarPlay in 2x and 3x scales, and for light and dark interfaces.

Your app icon in CarPlay should look similar to your app icon displayed on iPhone.

To set up your app icon, turn on CarPlay assets in Xcode and populate the CarPlay image wells.



Turn on CarPlay app icon assets



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)
CarPlay app icon	60pt x 60pt	180px x 180px	120px x 120px
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 need to know the CarPlay screen scale, 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).

Provide image resources that are display-ready, including light and dark interfaces. The best way to provide images for different appearances is by using asset catalogs. Each asset in an asset catalog represents different variations of the same image. Alternatively, you can use `UIImageAsset` to combine different variations of an image into a single package at runtime.

Build your 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 a static scene manifest in your `Info.plist` file. The following is an example of a 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.

Note that navigation apps also declare a scene for the CarPlay Dashboard and listen to different methods. See [CarPlay navigation apps](#) for details.

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 = CPLListItem(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 = CPLListSection(items: [item])
let listTemplate = CPLListTemplate(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: CPIInterfaceController) {

        let nowPlayingTemplate = CPNowPlayingTemplate.shared()

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

Play audio

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

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, they won't expect the FM radio to stop when they launch your app. The FM radio should continue to play until they explicitly choose another audio stream using your app. Don't activate your audio session at the time your app launches. Instead, wait until you actually need to play audio.

Work while iPhone is locked

CarPlay is frequently used while iPhone is in a locked state. Test your app thoroughly 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`.

CarPlay navigation apps

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 categories. Skip this section if you are not creating a navigation app.

Templates

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.

You need to be able to draw your map on vehicle screens with a variety of aspect ratios and resolutions. See [Simulator](#) for more information on testing with different display configurations.

The base view must be used exclusively to draw a map, and may not 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 the provided templates. Your app won't receive direct tap or drag events in the base view.

The base view can be used in navigation apps.



Base view

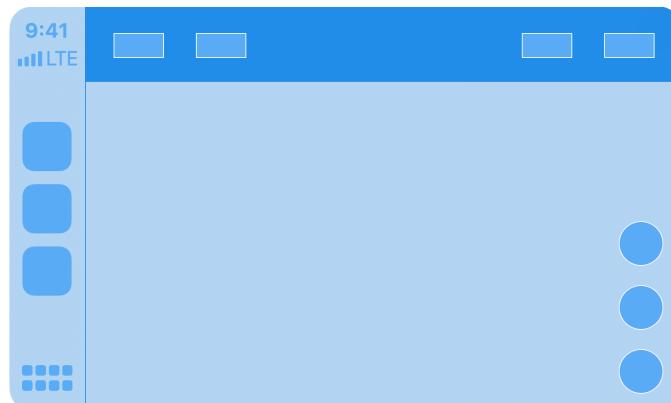
Map

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`.

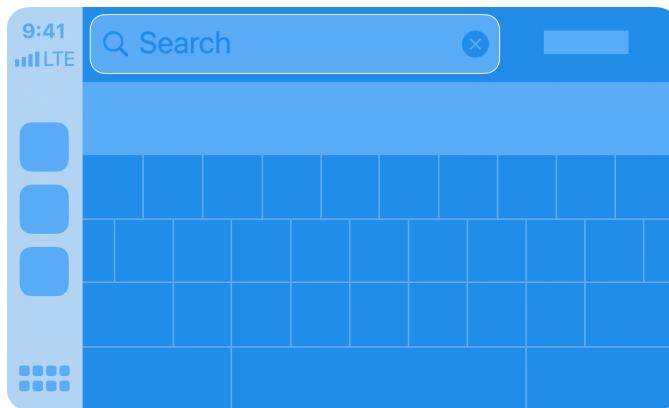
The base view can be used in navigation apps.



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 `CPLListItem` 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. The search template can be used in navigation apps.



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



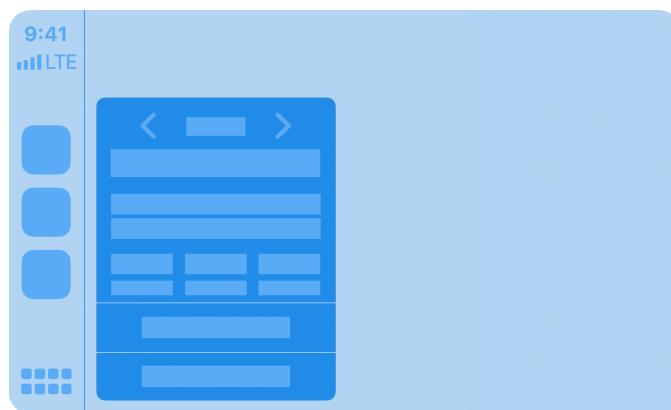
Voice control

Panels

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

Trip preview

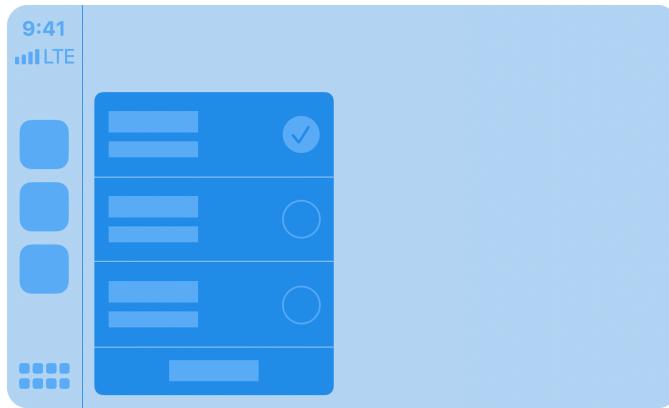
The trip preview panel allows users to choose from a list of up to 12 destinations, usually as the result of a destination search. When users preview a trip, show a visual representation of that trip in your base view.



Trip preview

Route choice

The route choice panel allows users to choose from a list of potential routes for a particular trip. 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

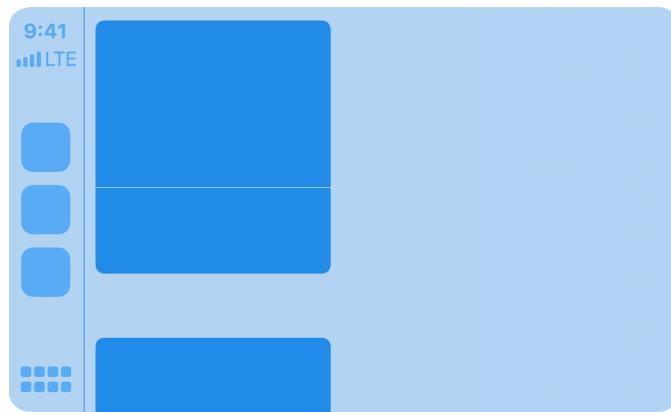
Guidance and trip estimate

The guidance and trip estimate panels displays upcoming maneuvers and trip estimates.

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

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

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



Guidance and trip estimate

Use the following guide when providing maneuver images. For maneuver symbols, provide variants for light and dark interfaces.

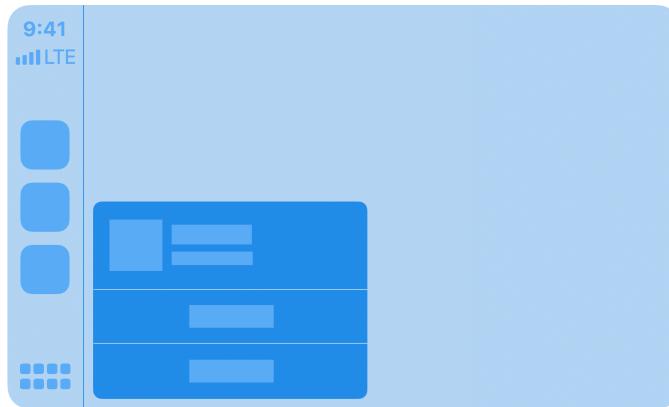
	Maximum size in points	Maximum size in pixels (3x)	Maximum size in pixels (2x)
First maneuver symbol (symbol and instruction on one line)	50pt x 50pt	150px x 150px	100px x 100px
First maneuver symbol (symbol and instruction on two lines)	120pt x 50pt	360px x 150px	240px x 100px
Second maneuver symbol (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

Navigation alerts provide 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 trigger a notification if your app is running in the background.

Provide information to the user using an image and a short text description. You may provide a primary and secondary action button. For example, the user could choose to maintain the current route, or take an alternate route.

You can also specify a duration for which the alert should be visible before it's automatically dismissed.



Navigation alert

Startup

CarPlay navigation apps can declare two CarPlay scenes, one for the main app window in CarPlay, and one for the CarPlay Dashboard. If you are using a scene manifest, extend it with information about the CarPlay Dashboard scene. See [Dashboard scene manifest](#) for details.

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 have 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 up to 12 nearby 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, you may present the routes as options for the user to choose from.

Show 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.

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)
```

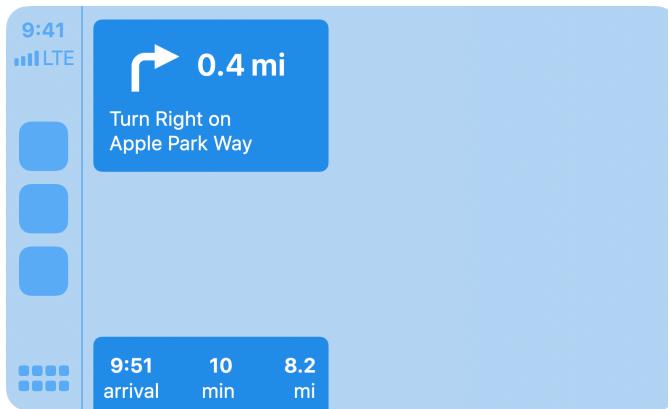
While you are calculating initial maneuvers, set the navigation session pause state to `CPTripPauseReasonLoading` so that CarPlay can display the correct state.

```
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.

Show 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, and estimates for remaining time and distance.



Show a maneuver in the route guidance panel

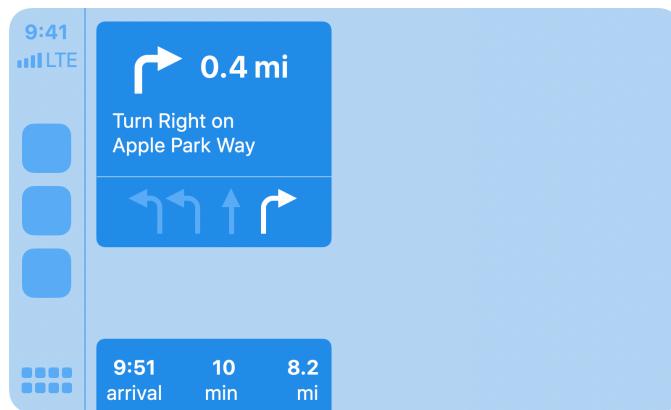
Symbol. If the maneuver has an associated symbol, such as a turn right arrow, provide an image using `symbolSet`. The symbol will be shown in the route guidance card and any related notifications. You must provide two image variants using `CPIImageSet`—one is used for rendering the symbol on light backgrounds,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

Maintain at least one upcoming turn in the maneuvers array at all times. 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.

[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

CarPlay Dashboard

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

`CPTemplateApplicationDashboardScene` is a `UIScene` subclass that CarPlay creates when it determines that your app should appear in CarPlay Dashboard.

`CPDashboardController` and `CPDashboardButton` let you manage controls that appear in CarPlay Dashboard.

Create a scene delegate

To begin, define a delegate for the dashboard scene just like you would for the main template application scene. This delegate conforms to `CPTemplateApplicationDashboardSceneDelegate` and will be given an instance of `CPDashboardController` and `UIWindow`.

Additionally, to support `CPTemplateApplicationDashboardScene`, include a new key in your Application Scene Manifest to declare support for CarPlay Dashboard: `CPSupportsDashboardNavigationScene` with a value of true.

Draw your CarPlay Dashboard content

Use the provided window to draw navigation related content for display in the CarPlay Dashboard, similar to the `CPWindow` given to the main template application scene. You can also provide two instances of `CPDashboardButton` to `CPDashboardController` that appear in the guidance card widget while your app is not actively navigating.

People can interact with your app through both the dashboard buttons as well as within your main app interface. When navigation begins in your app using `CPMapTemplate` and `CPNavigationSession`, CarPlay automatically displays maneuvers in the guidance widget. When navigation ends, it will revert to show your dashboard buttons.

Dashboard scene manifest

The following is an example of a scene manifest that supports the CarPlay Dashboard.

```
<key>UIApplicationSceneManifest</key>
<dict>
    <key>CPSupportsDashboardNavigationScene</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>
    </dict>
</dict>
```

```
<!-- For the CarPlay Dashboard scene -->
<key>CPTemplateApplicationDashboardSceneSessionRoleApplication</key>
<array>
<dict>
<key>UISceneClassName</key>
<string>CPTemplateDashboardScene</string>
<key>UISceneConfigurationName</key>
<string>CarPlay-Dashboard</string>
<key>UISceneDelegateClassName</key>
<string>MyAppCarPlayDashboardSceneDelegate</string>
</dict>
</array>
</dict>
</dict>
```

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.
Copyright © 2020 Apple Inc.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer or device for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-branded products.

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

Apple is a trademark of Apple Inc., registered in the U.S. and other countries.

APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT, ERROR OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

Some jurisdictions do not allow the exclusion of implied warranties or liability, so the above exclusion may not apply to you.