

[CarPlay](#) / Integrating CarPlay with your quick-ordering app

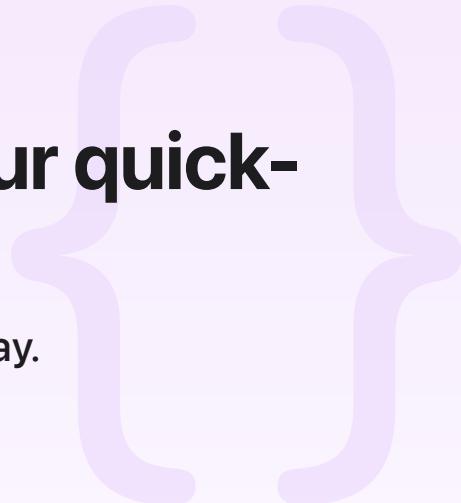
Sample Code

Integrating CarPlay with your quick-ordering app

Configure your food-ordering app to work with CarPlay.

[Download](#)

iOS 17.2+ | macOS 14.0+ | Xcode 15.4+



Overview

This sample code project demonstrates how to display custom ordering options in a vehicle using CarPlay. 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.

Configure the sample code project

CarPlay quick-ordering apps require a CarPlay quick-ordering entitlement, which you can request [here](#). After Apple grants the entitlement, follow these steps:

1. Log in to your account on the Apple Developer website and create a new provisioning profile that includes the CarPlay quick ordering-app entitlement.
2. Import the newly created provisioning profile into Xcode.
3. Create an `Entitlements.plist` file in the project, if you don't have one already.
4. Create a key for the CarPlay quick-ordering app entitlement as a Boolean. Make sure that the target project setting `CODE_SIGN_ENTITLEMENTS` has the path of the `Entitlements.plist` file.

Handle communication with CarPlay

After a device connects to CarPlay and the sample app launches, it sets a root template to display content onscreen. The system sets the root template on the `CPInterfaceController` when the app connects to CarPlay. In this sample, the root template is an instance of `CPTabBarTemplate` with a `CPPointOfInterestTemplate` as the template that initially displays.

```
func interfaceControllerDidConnect(_ interfaceController: CPInterfaceController, scene: CPScene) {
    MemoryLogger.shared.appendEvent("Connected to CarPlay window.")
    carplayInterfaceController = interfaceController
    carplayScene = scene
    carplayInterfaceController?.delegate = self
    sessionConfiguration = CPSessionConfiguration(delegate: self)
    locationManager.delegate = self
    requestLocation()
    setupMap()
}

func setupMap() {
    let pointOfInterestTemplate = CPPointOfInterestTemplate(
        title: "Hoagie Options",
        pointsOfInterest: [],
        selectedIndex: NSNotFound)
    pointOfInterestTemplate.pointOfInterestDelegate = self
    pointOfInterestTemplate.tabTitle = "Map"
    pointOfInterestTemplate.tabImage = UIImage(systemName: "car")!

    let tabTemplate = CPTabBarTemplate(templates: [pointOfInterestTemplate])

    carplayInterfaceController?.setRootTemplate(tabTemplate, animated: true, completion: { [weak self] in
        // Note: Ensure that 12 is the maximum POI locations that appear on the display
        self?.search(for: "Hoagies")
    })
}
```

Keep the map in focus

The sample uses `func pointOfInterestTemplate(_ aTemplate: CPPointOfInterestTemplate, didChangeMapRegion region: MKCoordinateRegion)` to keep the map in focus. The data that `CLLocationManager` provides for a given location can change as a person is moving. This means the results need to update as the map region changes.

```

extension TemplateManager: CPPointOfInterestTemplateDelegate {
    func pointOfInterestTemplate(_ aTemplate: CPPointOfInterestTemplate, didChangeMa
        MemoryLogger.shared.appendEvent("Region Changed: \(region).")
        // In your app, you need to update your search results when this triggers.
        boundingRegion = region
        search(for: "hoagies")
    }
}

```

After someone selects an item, options to place an order and open Maps for directions, or to call the point of interest, appear onscreen, depending on metadata availability.

```

// Make ordering the primary button.
let button = CPTextButton(title: "Order", textStyle: .normal, handler: { (button) i
    MemoryLogger.shared.appendEvent("Order tapped \(place).")
    self.showOrderTemplate(place: place)
})

place.primaryButton = button
// Try directions or a phone number as the secondary button.
if let address = place.summary,
    let encodedAddress = address.addingPercentEncoding(withAllowedCharacters: Charact
    let lon = place.location.placemark.location?.coordinate.longitude,
    let lat = place.location.placemark.location?.coordinate.latitude,
    let url = URL(string: "maps://?q=\(encodedAddress)&ll=\(lon),\(lat)") {
        place.secondaryButton = CPTextButton(title: "Directions", textStyle: .normal, ha
            MemoryLogger.shared.appendEvent("Opening Maps with \(address).")
            self.carplayScene?.open(url, options: nil, completionHandler: nil)
    })
} else if let phoneNumber = place.subtitle, let url = URL(string: "tel://" + phoneNu
    place.secondaryButton = CPTextButton(title: "Call", textStyle: .normal, handler:
        MemoryLogger.shared.appendEvent("Calling \(phoneNumber).")
        self.carplayScene?.open(url, options: nil, completionHandler: nil)
    ))
}

```

Because the sample relies on a person's location to provide relevant results, handle permission issues gracefully. The sample removes any presented view controllers and presents a message that the location isn't available.

```

func locationManagerDidChangeAuthorization(_ manager: CLLocationManager) {
    switch manager.authorizationStatus {

```

```

case .denied, .restricted, .notDetermined:
    let alert = CPAAlertTemplate(
        titleVariants: ["Please enable location services."],
        actions: [
            CPAAlertAction(
                title: "Ok",
                style: .default,
                handler: { [weak self] (action) in
                    self?.carplayInterfaceController?.setRootTemplate(
                        CPTabBarTemplate(templates: []), animated: false, completion: {
                            MemoryLogger.shared.appendEvent("Error setting root template")
                        })
                }
            )
        ]
    )

    // Check for a presented template and dismiss it for this important message.
    if carplayInterfaceController?.presentedTemplate != nil {
        dismissAlertAndPopToRootTemplate {
            self.carplayInterfaceController?.presentTemplate(alert, animated: false)
            self?.handleError(error, prependedMessage: "Error presenting \(alert.title)")
        }
    } else {
        carplayInterfaceController?.presentTemplate(alert, animated: false, completion: {
            self?.handleError(error, prependedMessage: "Error presenting \(alert.title)")
        })
    }
}

default:
    dismissAlertAndPopToRootTemplate {
        self.setupMap()
    }
    return
}
}

```

Provide updates

After a person places an order, the system starts a Live Activity to show the order's status. Live Activities don't display in CarPlay, but do provide a glanceable view on the person's Lock Screen to inform them about updates to their order.

```

MemoryLogger.shared.appendEvent("Placing Order")
do {

    // Simulate a scenario where a person using the app enters a tunnel without
    // place an order. The test API can't confirm the order in sendOrderToHQ.
    // The Live Activity starts manually.

    let attrs = OrderStatusAttributes(hoagieOrder: hoagieOrder)
    let initialState = OrderStatusAttributes.ContentState(
        isPickedUp: false,
        isReady: false,
        isPreparing: false,
        isConfirmed: true)

    try saveOrderState(state: initialState)

    MemoryLogger.shared.appendEvent("Starting Live Activity")
    OrderingService.service.orderActivity = try Activity.request(
        attributes: attrs,
        content: .init(state: initialState, staleDate: Date(timeIntervalSinceNow: 10)),
        pushType: .token
    )
    try await finalizeOrder(hoagieOrder: hoagieOrder)
} catch {
    throw OrderingError.errorOrdering
}

```

After the Live Activity is running, you need to create a listener for updates to the state of the activity and token changes. Your app can update Live Activities, but only in the foreground. If your app spends significant amounts of time in the background, such as in a quick-ordering app, you need to use notifications to provide updates to people using the app. The code example below shows one way to listen for updates to the Live Activity token. Your app doesn't need to support background updates, or use any UIApplication cycle methods. The system wakes the process that contains your Live Activity when the token changes and calls the attached block of code on the listener.

```

// For the purposes of this demonstration, hoagies are ready in 10 minutes or longer.
// Here, a push notification indicates whether an order is ready earlier.
// Spin off another thread to listen for updates.

Task { @MainActor in
    MemoryLogger.shared.appendEvent("Change Listener Task Started")
    for await change in activity.contentUpdates {
        MemoryLogger.shared.appendEvent("Content update change \(change.description)")
    }
}

```

```

        try saveOrderState(state: change.state)
        WidgetCenter.shared.reloadAllTimelines()
    }

Task { @MainActor in
    MemoryLogger.shared.appendEvent("State Listener Task Started")
    for await state in activity.activityStateUpdates {
        MemoryLogger.shared.appendEvent("Content update change \(state)")
        if state == .dismissed || state == .ended {
            await activity.end(nil, dismissalPolicy: .immediate)
            OrderingService.service.updateTokens[activity.id] = nil
        }
        WidgetCenter.shared.reloadAllTimelines()
    }
}

Task { @MainActor in
    MemoryLogger.shared.appendEvent("Push Token Update Listener Task Started")
    for await pushToken in activity.pushTokenUpdates {
        let pushTokenString = pushToken.reduce("") {
            $0 + String(format: "%02x", $1)
        }

        OrderingService.service.updateTokens[activity.id] = pushTokenString
        try await self.sendPushToken(hoagieOrder: hoagieOrder, pushTokenString)
    }
}

```

The sample includes a macOS target that provides a mock order status app. This is similar to what a service provider uses to convey updates to an order. The service app needs a JSON Web Token (JWT) to create push notifications for use with Live Activities. The following code example shows how the sample creates the JWT for use with the payload to send to Apple, which in turn sends a push notification to the associated device:

```

private static func createJWT() throws -> String {
    if TestHoagieData.hoagieDefaults.string(forKey: savedTokenKey) == nil {
        let symKey = try P256.Signing.PrivateKey(pemRepresentation: privateKey)
        let headerJSONData = try JSONEncoder().encode(Header())
        let headerBase64String = headerJSONData.urlSafeBase64EncodedString()
        let payloadJSONData = try JSONEncoder().encode(Payload())
        let payloadBase64String = payloadJSONData.urlSafeBase64EncodedString()
    }
}

```

```
let toSign = Data((headerBase64String + "." + payloadBase64String).utf8)
let signature = try symKey.signature(for: toSign)
let signatureBase64String = signature.rawRepresentation.urlSafeBase64Encoded
let token = [headerBase64String, payloadBase64String, signatureBase64String]
TestHoagieData.hoagieDefaults.set(Date.now, forKey: lastTokenCreationDate)
TestHoagieData.hoagieDefaults.set(token, forKey: savedTokenKey)
print(token)
return token
} else if
    let savedDate = TestHoagieData.hoagieDefaults.object(forKey: lastTokenCreationDate)
    Date.now.timeIntervalSince(savedDate) > TestHoagieData.tenMinutes {
    TestHoagieData.hoagieDefaults.set(nil, forKey: lastTokenCreationDate)
    TestHoagieData.hoagieDefaults.set(nil, forKey: savedTokenKey)
    return try createJWT()
} else if let token = TestHoagieData.hoagieDefaults.string(forKey: savedTokenKey)
    print(token)
    return token
} else {
    fatalError()
}
}
```

See Also

Location and Information

class **CPPointOfInterestTemplate**

A template that displays a map with selectable points of interest.

class **CPIInformationTemplate**

A template that provides information for a point of interest, food order, parking location, or charging location.

class **CPTextButton**

A button that displays a stylized title.