Sample Code

Offering Apple Pay in Your App

Collect payments with iPhone and Apple Watch using Apple Pay.

Download

iOS 15.0+ | iPadOS 15.0+ | watchOS 8.0+ | Xcode 13.0+

Overview

This sample shows the implementation of an integrated Apple Pay eCommerce experience across iOS and watchOS. The sample app demonstrates how to use the Apple Pay button, display the Apple Pay payment sheet, make payment requests, and accept coupon codes.

The sample ticket booking app implements buying a ticket using Apple Pay in:

- An iOS app.
- A watchOS app.

A shared `PaymentHandler` class handles payment in each of the apps.

> **Note**
>
> This sample code project is associated with WWDC21 session 10092: What's New in Wallet and Apple Pay.

Configure the Sample Code Project

Test Apple Pay payments with this sample by configuring the bundle identifiers and Apple Pay configuration items in Xcode. Doing this requires an Apple developer account. Before building the app, complete these four steps:

1. Change the bundle ID for each target so that it's unique for your developer account; change `example` in the bundle ID to something that represents you or your organization.
2. In the build settings, update the value of the user-defined `OFFERING_APPLE_PAY_BUNDLE_PREFIX` setting to match the prefix of the bundle IDs you changed in step 1. For example, if you changed `example` in each bundle ID to your organization name, change `example` in `OFFERING_APPLE_PAY_BUNDLE_PREFIX` to the same organization name. Xcode configures the required merchant ID for Apple Pay in each target when you build the project.
3. Set up the Apple Pay Merchant Identity and Apple Pay Payment Processing certificates. For more information on setting up a merchant identity and processing certificates, see Setting Up Apple Pay.
4. Set the signing option for the iOS app to "Automatically manage signing."

Running this app on an iPhone or Apple Watch requires an Apple Pay card. Running in Simulator doesn't require a card.

> **Note**
>
> Not all Apple Pay features are supported in the iOS simulator. Testing Apple Pay is unsupported in the watchOS simulator.

For more information about processing an Apple Pay payment using a payment platform or merchant bank, see An easier way to pay within apps and websites. To set up your sandbox environment for testing, see Sandbox Testing.

Add the Apple Pay Button

Apple Pay includes pre-built buttons to start a payment interaction, or to set up payment methods. The iOS app displays the payment button if the device can make payments, and it contains at least one payment card; otherwise it displays the button to add a payment.

This sample checks for the ability to make payments using `canMakePayments()`, and checks for available payment cards using `canMakePayments(usingNetworks:)`. Both of these methods are part of `PKPaymentAuthorizationController`.

```
static let supportedNetworks: [PKPaymentNetwork] = [
    .amex,
    .discover,
    .masterCard,
    .visa
]
```

```
class func applePayStatus() -> (canMakePayments: Bool, canSetupCards: Bool) {
    return (PKPaymentAuthorizationController.canMakePayments(),
            PKPaymentAuthorizationController.canMakePayments(usingNetworks: supporte
}
```

The iOS app displays the payment button by adding an instance of PKPaymentButton.

> **Note**
>
> The sample app doesn't display the add button if a device can't accept payments due to hardware limitations, parental controls, or any other reasons.

```
let result = PaymentHandler.applePayStatus()
var button: UIButton?

if result.canMakePayments {
    button = PKPaymentButton(paymentButtonType: .book, paymentButtonStyle: .black)
    button?.addTarget(self, action: #selector(ViewController.payPressed), for: .touc
} else if result.canSetupCards {
    button = PKPaymentButton(paymentButtonType: .setUp, paymentButtonStyle: .black)
    button?.addTarget(self, action: #selector(ViewController.setupPressed), for: .to
}

if let applePayButton = button {
    let constraints = [
        applePayButton.centerXAnchor.constraint(equalTo: applePayView.centerXAnchor)
        applePayButton.centerYAnchor.constraint(equalTo: applePayView.centerYAnchor)
    ]
    applePayButton.translatesAutoresizingMaskIntoConstraints = false
    applePayView.addSubview(applePayButton)
    NSLayoutConstraint.activate(constraints)
}
```

The watchOS app adds the button to the storyboard as an instance of WKInterfacePaymentButton.

Define the Shipping Methods

The app defines two shipping methods: delivery with estimated shipping dates and on-site collection. The payment sheet displays the delivery information for the chosen shipping method,

including estimated delivery dates. Configuring the dates requires a calendar, start date components, and end date components

```
func shippingMethodCalculator() -> [PKShippingMethod] {
    // Calculate the pickup date.

    let today = Date()
    let calendar = Calendar.current

    let shippingStart = calendar.date(byAdding: .day, value: 3, to: today)!
    let shippingEnd = calendar.date(byAdding: .day, value: 5, to: today)!

    let startComponents = calendar.dateComponents([.calendar, .year, .month, .day],
    let endComponents = calendar.dateComponents([.calendar, .year, .month, .day], fr

    let shippingDelivery = PKShippingMethod(label: "Delivery", amount: NSDecimalNumb
    shippingDelivery.dateComponentsRange = PKDateComponentsRange(start: startCompone
    shippingDelivery.detail = "Ticket sent to you address"
    shippingDelivery.identifier = "DELIVERY"

    let shippingCollection = PKShippingMethod(label: "Collection", amount: NSDecimal
    shippingCollection.detail = "Collect ticket at festival"
    shippingCollection.identifier = "COLLECTION"

    return [shippingDelivery, shippingCollection]
}
```

Start the Payment Process

Both iOS and watchOS implementations start the payment process by calling the `startPayment` method of the shared `PaymentHandler`. Updates to the payment sheet use the completion handlers implemented by both apps. The `startPayment` method stores the completion handlers because the Apple Pay functionality is asynchronous; and then the method creates an array of `PKPaymentSummaryItem` to display the charges on the payment sheet.

```
let ticket = PKPaymentSummaryItem(label: "Festival Entry", amount: NSDecimalNumber(s
let tax = PKPaymentSummaryItem(label: "Tax", amount: NSDecimalNumber(string: "1.00")
let total = PKPaymentSummaryItem(label: "Total", amount: NSDecimalNumber(string: "10
paymentSummaryItems = [ticket, tax, total]
```

The app configures a `PKPaymentRequest` using the list of payment items and other details.

```
let paymentRequest = PKPaymentRequest()
paymentRequest.paymentSummaryItems = paymentSummaryItems
paymentRequest.merchantIdentifier = Configuration.Merchant.identifier
paymentRequest.merchantCapabilities = .capability3DS
paymentRequest.countryCode = "US"
paymentRequest.currencyCode = "USD"
paymentRequest.supportedNetworks = PaymentHandler.supportedNetworks
paymentRequest.shippingType = .delivery
paymentRequest.shippingMethods = shippingMethodCalculator()
paymentRequest.requiredShippingContactFields = [.name, .postalAddress]
#if !os(watchOS)
paymentRequest.supportsCouponCode = true
#endif
```

Next the app displays the payment sheet by calling `PKPaymentAuthorizationController` with the payment request. Both apps present the payment sheet using `present(completion:)`.

The payment sheet handles all user interactions, including payment confirmation. It requests updates using the completion handlers stored by the `startPayment` method when a user updates the sheet.

```
paymentController = PKPaymentAuthorizationController(paymentRequest: paymentRequest)
paymentController?.delegate = self
paymentController?.present(completion: { (presented: Bool) in
    if presented {
        debugPrint("Presented payment controller")
    } else {
        debugPrint("Failed to present payment controller")
        self.completionHandler(false)
    }
})
```

Respond to Coupon Code Entry

The `PaymentHandler` class handles coupons by implementing the `PKPaymentAuthorizationControllerDelegate` protocol method `paymentAuthorizationController(_:didSelectPaymentMethod:handler:)`.

After the user enters an accepted coupon code, the method adds a new `PKPaymentSummaryItem` displaying the discount, and adjusts the `PKPaymentSummaryItem` with the discounted total.

> **Note**
>
> This method is wrapped in a conditional compilation flag as watchOS 8 doesn't support coupon codes.

```
#if !os(watchOS)

func paymentAuthorizationController(_ controller: PKPaymentAuthorizationController,
                                    didChangeCouponCode couponCode: String,
                                    handler completion: @escaping (PKPaymentRequestC
    // The `didChangeCouponCode` delegate method allows you to make changes when the

    func applyDiscount(items: [PKPaymentSummaryItem]) -> [PKPaymentSummaryItem] {
        let tickets = items.first!
        let couponDiscountItem = PKPaymentSummaryItem(label: "Coupon Code Applied",
        let updatedTax = PKPaymentSummaryItem(label: "Tax", amount: NSDecimalNumber(
        let updatedTotal = PKPaymentSummaryItem(label: "Total", amount: NSDecimalNum
        let discountedItems = [tickets, couponDiscountItem, updatedTax, updatedTotal
        return discountedItems
    }

    if couponCode.uppercased() == "FESTIVAL" {
        // If the coupon code is valid, update the summary items.
        let couponCodeSummaryItems = applyDiscount(items: paymentSummaryItems)
        completion(PKPaymentRequestCouponCodeUpdate(paymentSummaryItems: applyDiscou
        return
    } else if couponCode.isEmpty {
        // If the user doesn't enter a code, return the current payment summary item
        completion(PKPaymentRequestCouponCodeUpdate(paymentSummaryItems: paymentSumm
        return
    } else {
        // If the user enters a code, but it's not valid, we can display an error.
        let couponError = PKPaymentRequest.paymentCouponCodeInvalidError(localizedDe
        completion(PKPaymentRequestCouponCodeUpdate(errors: [couponError], paymentSu
        return
    }
}

#endif
```

Handle Payment Success or Failure

When the user authorizes the payment, the system calls the paymentAuthorizationController(_:didAuthorizePayment:handler:) method of the PKPaymentAuthorizationControllerDelegate protocol. Your handler confirms that the shipping address meets the criteria needed, and then calls the completion handler to report success or failure of the payment.

The sample code contains a comment at the place you add code for processing the payment.

```
func paymentAuthorizationController(_ controller: PKPaymentAuthorizationController,

    // Perform basic validation on the provided contact information.
    var errors = [Error]()
    var status = PKPaymentAuthorizationStatus.success
    if payment.shippingContact?.postalAddress?.isoCountryCode != "US" {
        let pickupError = PKPaymentRequest.paymentShippingAddressUnserviceableError(
        let countryError = PKPaymentRequest.paymentShippingAddressInvalidError(withK
        errors.append(pickupError)
        errors.append(countryError)
        status = .failure
    } else {
        // Send the payment token to your server or payment provider to process here
        // Once processed, return an appropriate status in the completion handler (s
    }

    self.paymentStatus = status
    completion(PKPaymentAuthorizationResult(status: status, errors: errors))
}
```

Once the sample app calls the completion handler in the paymentAuthorizationController(_:didAuthorizePayment:handler:) method, Apple Pay tells the app it can dismiss the payment sheet by calling paymentAuthorizationControllerDidFinish(_:). iOS and watchOS both handle dismissing the sheet as appropriate for each platform. In the iOS app, if payment succeeds the completion handler performs a segue to display a new view controller. If the payment fails, it remains on the payment sheet so the user can attempt payment with a different card or correct any issues.

```
@objc func payPressed(sender: AnyObject) {
    paymentHandler.startPayment() { (success) in
        if success {
            self.performSegue(withIdentifier: "Confirmation", sender: self)
```

```
            }
        }
}
```

See Also

Apple Pay setup

- Setting up Apple Pay

  Fulfill the requirements to provide Apple Pay as a payment option on your website or in your app.

- Complying with regional regulations

  Check regional regulations for possible requirements for your Apple Pay-based implementation.