Article

Adding support for Tap to Pay on iPhone to your app

Configure your app to use Tap to Pay on iPhone to read contactless payment cards.

Overview

Tap to Pay on iPhone lets an app use the person's iPhone to read contactless payment cards without any additional hardware. Merchants use your app to request payments from customers, and process the financial transactions using the back-end payment services you offer. Your app uses the ProximityReader framework to read the payment method someone presents. You then pass that payment method and transaction details to the payment service provider you use to handle financial transactions.

> **Important**
>
> Apps must have the appropriate entitlements to use Tap to Pay on iPhone. For more information, see Setting up Tap to Pay on iPhone.

The region or payment service provider may require a specific iOS version for merchants using Tap to Pay on iPhone. Make sure merchants are aware of any restrictions as part of your app materials.

- For a list of supported regions and payment service providers, see https://developer.apple.com/tap-to-pay/#regions.
- For additional merchant-related information, see Frequently asked questions for Tap to Pay on iPhone.

Verify the current device supports Tap to Pay on iPhone

Before you attempt to use Tap to Pay on iPhone, verify that the current device supports the feature. Tap to Pay on iPhone is available only on iPhone, starting with iPhone XS. To determine if the feature is available on the current device, check the value of the `isSupported` static property before you do anything else. If the feature is available, create a `PaymentCardReader` object to manage further interactions and to start sessions to read payment cards.

The following example shows a custom type that allocates a `PaymentCardReader` object only if the current device supports Tap to Pay on iPhone. Other methods of this type throw errors if the reader object isn't available.

```
public class MyPaymentProcessor {

    var reader: PaymentCardReader?

    init() {
        guard PaymentCardReader.isSupported else {
            // This device doesn't support Tap to Pay on iPhone.
            return
        }
        reader = PaymentCardReader()
    }
}
```

Display the Terms and Conditions interface to the merchant

The first time a merchant runs your app, present an interface for them to accept the Terms and Conditions for Tap to Pay on iPhone. Accepting the terms and conditions registers that merchant with Tap to Pay on iPhone, and allows them to begin using the feature. Merchants need to accept the terms and conditions only once for a given merchant identifier, even if they use the feature on a different device.

To display the Terms and Conditions for Tap to Pay on iPhone, get a JSON web token (JWT) from your payment service provider that includes the merchant's unique identifier. Store the token in a secure location so you can access it later. Pass the token to the `linkAccount(using:)` method of a `PaymentCardReader` object. This method presents a system sheet that displays the Terms and Conditions for Tap to Pay for iPhone.

The following code extends the `MyPaymentProcessor` class from the previous example with an additional function to present the terms and conditions:

```
extension MyPaymentProcessor {

    public func presentTermsAndConditions(tokenData: String) async throws {
        let token = PaymentCardReader.Token(rawValue: tokenData)

        // Display the UI only if the merchant hasn't accepted the terms and conditi
        if !reader.isAccountLinked(using:token) {
            // Confirm that the user is an admin. Otherwise, display a message that
            // states only admins may accept the Terms and Conditions.
            do {
                try await reader?.linkAccount(using: token)
            } catch {
                // Handle any errors that occur during linking.
            }
        }
    }
}
```

Create a session to read payment cards from the current device

Each time your app launches, create a `PaymentCardReaderSession` to handle the reading of payment card information. You create this session using the `prepare(using:)` method of your `PaymentCardReader` object. The initial configuration of the device can take up to two minutes, but subsequent requests typically take only a few seconds. The `events` property provides an asynchronous stream of events that you can use to display a progress UI to the merchant.

The following example configures a task to monitor progress events and then calls the `prepare(using:)` method to configure the device. After the successful configuration of the device, the `prepare(using:)` method returns a `PaymentCardReaderSession` object that you can use to start reading payment cards. If an error occurs, the method throws a `PaymentCardReaderError` for you to handle.

```
extension MyPaymentProcessor {

    var session: PaymentCardReaderSession?

    public func configureDevice(tokenData: String) async throws {
        guard let reader = self.reader else {
            return
        }
```

```
        guard let token = PaymentCardReader.Token(rawValue: tokenData) else {
            return
        }

        let events = reader.events
        do {

            // Create a task to monitor progress events.
            Task { @MainActor in
                for await event in events {
                    if case .updateProgress = event {
                        // Update the UI with the current progress value.
                    }
                }
            }

            // Prepare the device.
            session = try await reader.prepare(using: token)
        } catch {
            // Handle any errors that occur during preparation.
            // (see PaymentCardReaderError).
        }
    }
}
```

If the merchant closes your app while configuration is in progress, the `prepare(using:)` method throws a `PaymentCardReaderError.serviceConnectionError` error, but configuration continues in the background. When your app returns to the foreground, call `prepare(using:)` again to finish the previous configuration process and retrieve a new session object.

Important

When your app goes into the background, the current session ends. When that happens, set any variables that contain the session to `nil`. When your app returns to the foreground, call `prepare(using:)` every time to create a new session.

Present the UI to request someone's payment details

Once you have a `PaymentCardReaderSession` you can use it to display a sheet to the merchant's customer asking them for their payment method. The sheet displays a standard system payment UI and waits for the customer to present their card or other payment method to

the merchant's iPhone. As soon as the iPhone collects the payment information, the sheet closes and the session returns a `PaymentCardReadResult` to your app. If the sheet wasn't able to collect a payment information, it returns a `PaymentCardReaderSession.ReadError` to let you know that a problem occurred. Make sure your app handles the different types of errors that can occur for transactions.

The `PaymentCardReadResult` structure contains important information that your payment service provider needs to complete the transaction. For each transaction, send the encrypted data in the `paymentCardData` property to your payment service provider. Send this data regardless of whether or not the transaction completed successfully. The encrypted data is valid for only 60 seconds, so send it immediately to your provider.

The following example shows a template for a function that initiates a payment request for a session. The method creates a new `PaymentCardTransactionRequest` with the transaction amount and uses the session to read the customer's payment card. The method also monitors card reader events asynchronously and provides an appropriate response. For example, you might use events for your own analytics or to take appropriate actions.

```
extension MyPaymentProcessor {

    public func readCard(for amount: Decimal) async throws {
        guard let reader else { return }

        let request = PaymentCardTransactionRequest(amount: amount,
                                                    currencyCode: "USD",
                                                    for: .purchase)
        let events = reader.events
        do {
            Task {
                for await event in events {
                    // Handle events that happen while the sheet is up.
                }
            }

            // Present the payment interface and wait for the result.
            let result = try await session?.readPaymentCard(request)

            // Send result.paymentCardData to your payment service provider.
        } catch {
            // Handle any errors that occur during read
            // (see PaymentCardReaderSession.ReadError).
        }
    }
```

```
}
```

Catch any errors that happen during a payment read operation and take appropriate action. For more information about the errors that can occur, see `PaymentCardReaderSession.ReadError`.

Capture PIN information as part of a transaction

Some cards require the customer to enter an associated personal identification number (PIN) to complete a transaction. If the current card requires a PIN, the system UI automatically asks the person to enter it after reading the card. After the person enters the PIN and completes the transaction, the session returns a `PaymentCardReadResult` structure that contains the encrypted PIN data along with the rest of the transaction data.

A card issuer can also require a person to enter a PIN to complete a transaction. After reading the card and sending the transaction data to your payment service provider, you might get a subsequent request for the person's PIN data. Call the `capturePIN(using:cardReaderTransactionID:)` method to display a new sheet to ask for the person's PIN. Upon successful completion of this method, the session returns a `PaymentCardReadResult` with both the encrypted payment card data and the encrypted PIN data.

> **Important**
>
> After you read a person's payment card data, you have only 55 seconds to call `capturePIN(using:cardReaderTransactionID:)` to capture the person's PIN. If you don't call the method within this timeframe, the method returns `PaymentCardReaderSession.ReadError.pinNotAllowed`.

If a read operation requires a separate PIN capture, the payment service provider must provide a secure PIN token for that specific operation. Use this token to create a `PaymentCardReaderSession.PINToken` structure and display the capture UI. The following method captures the PIN for a previous read operation.

```
extension MyPaymentProcessor {

    public func capturePIN(for previousData: PaymentCardReadResult, with rawPINToker
        guard let reader else { return }

        let transactionId = previousData.id
        let token = PaymentCardReaderSession.PINToken(rawValue: rawPINTokenFromPSP)

        let events = reader.events
```

```
        do {
            Task {
                for await event in events {
                    // Handle events that happen while the sheet is up.
                }
            }

            // Display the PIN capture interface.
            let result = try await session?.capturePIN(using: token, cardReaderTrans

            // Send result.paymentCardData to your payment service provider.
        } catch {
            // Handle any errors that occur during PIN capture
            // (see PaymentCardReaderSession.ReadError).
        }
    }
}
```

Important

When capturing a PIN from a merchant's customer, do not display any correlatable data in an unsecure manner. A nonexhaustive list of correlatable data includes: phone numbers, email addresses, last 4 digits of payment cards with the expiry date, and so on. If you must display any correlatable data during PIN capture, the best practice is to show only a portion of the data. For example, show only the last 4 digits of the person's phone number: XXX-XXX-1234.

Turn off card reader functionality when not in use

If payment processing is one of several features of your app, you can turn off the card reader functionality when not in use. The simplest way to turn off this functionality is to deallocate the `PaymentCardReader` and `PaymentCardReaderSession` objects you currently have. The following method sets the `reader` and `session` variables of the previously defined `MyPaymentProcessor` class to `nil`:

```
extension MyPaymentProcessor {

    public func cleanup() {
        session = nil
        reader = nil
    }
}
```

When you delete the reader and session objects, the system clears out the current session information but retains some system-level details and configuration data. Creating a new reader and session later requires a network connection, because the reader requires information on the server to create any new sessions.

See Also

Payment card reader

Setting up Tap to Pay on iPhone

Request and configure the required entitlement to support Tap to Pay on iPhone.

`class` `PaymentCardReader`

An object you use to configure Tap to Pay on iPhone on the current device.

`class` `PaymentCardReaderSession`

The object you use to start reading a contactless payment or loyalty card.